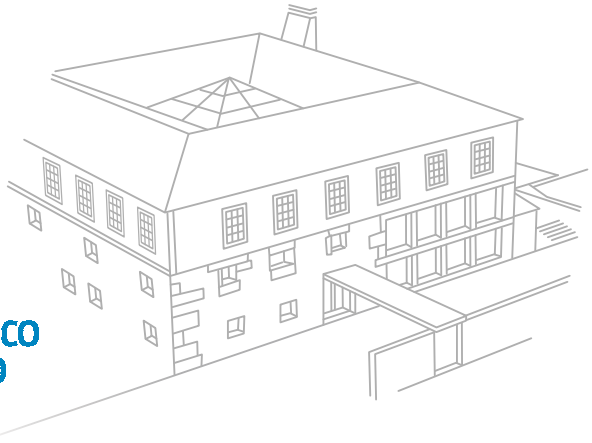

ESTGF | POLITÉCNICO DO PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Ferramentas para optimização da solução NGIN da PT Inovação

DESIGNAÇÃO DO MESTRADO
Mestrado em Engenharia Informática

AUTOR
Vítor Emanuel Moreira de Castro

ORIENTADOR(ES)
Carla Sofia Gonçalves Pereira

ANO
2010

Agradecimentos

Estando muito perto de concluir mais uma etapa do meu percurso académico, seria injusto não expressar os meus sinceros agradecimentos a todos aqueles que permitiram que o presente trabalho fosse concretizado com sucesso. Em particular gostaria de agradecer:

A toda a minha família, em especial aos meus pais e avós que, cada um da sua forma e ao seu jeito me deram educação, ajuda e a confiança necessária nos momentos mais difíceis.

À minha namorada, Mónica, pela compreensão, apoio, ajuda e força que me tem dado ao longo destes anos, bem como na ajuda da revisão deste trabalho.

À entidade onde desenvolvi este trabalho, PT Inovação, pela oportunidade dada e por acreditar que o meu contributo seria importante para realizar os projectos que compõem este trabalho, e em especial aos meus colegas de equipa.

À Professora Carla Pereira, minha orientadora, a ajuda que me deu para realizar este trabalho tanto a nível científico como psicológico, a paciência que teve quando alguns problemas foram encontrados e a motivação que sempre transmitiu, pois sem a sua ajuda e conhecimentos não seria possível cumprir os objectivos.

Ao meu companheiro de aventura, o meu amigo Tiago Areias, por toda a ajuda dada e por me acompanhar em mais uma etapa da minha vida.

A todos o meu muito obrigado!

Dedicatória

Ao meu avô, que é o meu herói e o meu exemplo de vida.

Resumo

Ter um produto de *software* fiável, sem erros, robusto e com uma utilização amigável, é uma premissa essencial para um produto de qualidade. Foi essa a intenção deste trabalho de mestrado com solução NGIN da PT Inovação.

Devido à constante disponibilização de novos serviços pela solução NGIN aliada ao grande número de configurações subjacentes aos seus serviços, surge a necessidade de criação de ferramentas que auxiliem essas mesmas configurações. Hoje, no contexto da plataforma NGIN, as aplicações de configuração constituem uma enorme vantagem face à concorrência, proporcionando a disponibilização de novos serviços e configurações em tempo cada vez mais reduzido e de uma forma cada vez mais fiável e robusta.

Com uma solução de uma enorme utilização e com um comportamento que tem de ser "infalível", são necessárias cada vez mais formas de validarem e avaliarem os desenvolvimentos feitos sobre a mesma. Não é espectável que um produto de *software* chegue ao cliente com problemas, sendo assim a procura de problemas e erros uma obrigatoriedade cada vez maior para as empresas que desenvolvem *software*.

Existindo a plena noção desta importância, este trabalho incide no desenvolvimento de ferramentas específicas. Se num primeiro caso o objectivo passa por testar, validar e avaliar parte do *software* desenvolvido, ou seja, desenvolver uma ferramenta de automatização de testes, a outra ferramenta tem como objectivo manipular e configurar as entidades e regras de negócio que compõem a solução, ou seja, criar um catálogo de produtos/serviços da solução NGIN.

Estas duas ferramentas têm como ponto central a optimização, a fiabilidade, a simplicidade de manipulação e configuração da solução NGIN.

Palavras-chave: NGIN, ferramenta de automatização de testes, catálogo de produtos/serviços

Abstract

Having a reliable, no error, robust and user friendly software product, is an essential premise for a quality product. That was the intention of this master thesis with NGIN of PT Inovação.

The constant availability of new services offered by NGIN solution, allied to the great number of settings underlie its services, there is the need to create tools to help these same settings. Today, in the context of NGIN platform, the configuration applications have a huge advantage against the competition, providing the availability of new services and configurations in time becoming a smaller and ever more reliable and robust.

With a solution of a massive use and a behavior that has to be "infallible", it's necessary more ways to validate and assess the developments made on it. It is not expected that a software product reaches the customer with problems, so the search for problems and mistakes is a growing requirement for companies that develop software.

Given the importance of these tools, this work focuses on the development of specific tools. If in a first case the objective is to test, validate and evaluate part of the software developed, ie developing a automation testing tool. The other one is a tool designed to manipulate and configure the business rules and entities that comprise the solution, ie create a catalog of products / services NGIN.

These two tools have as their central point the optimization, reliability, ease of handling and configuration of NGIN.

Keywords: NGIN, automation testing tool, a catalog of products/services.

Glossário de termos e abreviaturas

NGIN	*Next Generation Intelligent Network*
PT	Portugal Telecom
INS	*Intelligent Network Services*
SLTF	*Service Logic Telecommunications Solução*
PL/SQL	*Procedural Language/Structured Query Language*
CM	*Configuration Manager*
GWT	*Google Web Toolkit*
SS7	*Signaling System #7*
VoIP	*Voice Over IP*
IMS	*IP Multimedia Subsystem*
SIGTRAN	*Signaling Transporting*
SIP	*Session Initiation Protocol*
IETF	*Internet Engineering Task Force*
SQL	*Structured Query Language*
PCA	*Package Configuration Application*
XML	*eXtensible Markup Language*
W3C	*World Wide Web Consortium*
JVM	*Java Virtual Machine*
API	*Application Programming Interface*
HTML	*HyperText Markup Language*
CSS	*Cascading Style Sheet*
JRE	*Java Runtime Environment*
J2SE	*Java 2 Standard Edition*
J2EE	*Java 2 Enterprise Edition*
GWT-RPC	*Google Web Toolkit – Remote Procedure Call*
HTTP	*Hypertext Transfer Protocol*
ETL	*Extract Transform Load*
DAO	*Data Access Object*

ORM	*Object relational mapping*
VV&T	Verificação, validação e teste
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*International Organization for Standardization*
SQA	*Software Quality Assurance*
CMMi	*Capability Maturity Model Integration*
CMM	*Capability Maturity Model*
SEI	*Software Engineering Institute*
SW-CMM	*CMM for Software*
IEC	*International Electrotechnical Commission*
PMBOK	*Project Management Body of Knowledge*
CMMI-DEV	*Capability Maturity Model Integration for Development*
NGOSS	*New Generation Operations Systems and Software*
eTOM	*enhanced Telecom Operations Map*
TAM	*Telecom Applications Map*
TM	*Telecommunication Management*
OSS	*Operation Support Systems*
BSS	*Business support systems*
SANRR	*Scope, Analyze, Normalize, Rationalize e Rectify*
SID	*Shared Information and Data Model*
ITIL	*Information Technology Infrastructure Library*
TOGAF	*The Open Group Architecture Framework*
UML	*Unified Modeling Language*
FAB	*Fulfillment, Assurance, & Billing*
OSR	*Operational Support Readiness*
CRM	*Customer relationship management*
PIN	*Personal identification number*
IVR	*Interactive voice response*
GUI	*Graphical user interface*
CDM	*Canonical Data Model*

SOA	*Service-oriented architecture*
SOAP	*Simple Object Access Protocol*
MVC	*Model View Controller*
LCV	Lógica de Controlo de Versões
POJO	*Plain Old Java Object*
XSD	*XML Schema Definition*
DTO	*Data Transfer Objects*
AOP	*Aspect-oriented programming*
SCA	Sistema de controlo de acessos

Índice

Índice de Figuras

1. Introdução

Neste capítulo introdutório, será descrito o contexto e a relevância deste trabalho, são apresentadas as motivações e os objectivos a atingir. A estrutura desta tese é aqui também descrita.

1.1. Contextualização

Este projecto está enquadrado na solução NGIN (*Next Generation Intelligent Network*), da PT (Portugal Telecom) Inovação. A solução está baseada no conceito de rede inteligente e tem evoluído ao longo da última década de modo a satisfazer não só as necessidades do negócio nos mercados onde é utilizada, mas também a própria evolução tecnológica ao nível das redes e dos serviços. A NGIN é bastante complexa e de uma enorme dimensão, sendo que o trabalho aqui apresentado incide exclusivamente sobre um subsistema desta solução.

Este projecto aborda os problemas e algumas lacunas existentes na utilização da solução NGIN, apresentando possíveis soluções para optimizar e tornar cada vez mais fiável a solução.

Apresentando primeiramente a empresa que possibilitou a elaboração deste trabalho, a PT Inovação é uma empresa do grupo PT vocacionada para o desenvolvimento de novos serviços e soluções de telecomunicações, contribuindo, desta forma, para o aumento da competitividade e liderança do grupo PT nos sectores e mercados onde actua. A solução NGIN está espalhada pelos 4 cantos do mundo e é usada por mais de 80 milhões de clientes. A figura 1.1 apresenta os mercados onde a solução se encontra implementada [1].

Figura 1.1 – Clientes com solução NGIN [1]

Toda esta solução assenta sobre um conjunto de produtos compostos por um conjunto de subsistemas. Não serão apresentados todos os produtos exaustivamente, apenas será abordado o produto INS *(Intelligent Network Services)* e especificamente um dos seus subsistemas, o subsistema SLTF

(Service Logic Telecommunications Solução). O SLTF é composto pelos componentes NGIN Care[1], NGIN Core[2], *Configuration Manager* e a suite genérica[3]. No capítulo seguinte será detalhado esta relação.

Descrevendo o que foi feito no projecto, foram desenvolvidas duas ferramentas que actuam sobre a solução NGIN. A primeira ferramenta, uma ferramenta de testes e validação de *software* denominada de *TestBeds* vai actuar directa e exclusivamente na suite genérica. Avaliados os testes e mediante estes serem bem sucedidos, a aplicação valida que os módulos desenvolvidos estão prontos a serem usados. É bastante importante que exista esta validação pois entregar um produto com erros é muito prejudicial para a empresa. Se tudo correr bem como é desejado e espectável o componente suite genérica fica disponível para o cliente.

O segundo projecto surge da complexidade da solução NGIN. Um produto complexo como este é, necessita de uma ferramenta que apoie o seu manuseamento e a sua configuração. A ferramenta *Configuration Manager* como foi baptizada, é composta por um conjunto de módulos que cooperam entre si, com o objectivo de permitir a definição de entidades e regras de negócio que determinam o comportamento do sistema NGIN. Na prática o CM vai permitir configurar ofertas comerciais que as operadoras disponibilizam para os seus clientes. Desta forma, pretende-se evoluir de uma ferramenta pouco amigável e que configurava parte do sistema NGIN (apenas o NGIN CORE) para uma ferramenta que oriente o utilizador por passos guiados através das configurações, tanto do CARE como do CORE, usando um mecanismo de navegação pelo mapa de dependências entre as configurações, possibilitando também uma edição massiva de configurações.

Estas duas ferramentas têm como ponto central a optimização, o enriquecimento, a fiabilidade da solução NGIN. Se nas *TestBeds* o grande objectivo é que o cliente receba uma solução sem erros, cada vez mais fiável, o *Configuration Manager* vai usar essa solução cada vez mais robusta e fiável para permitir aos clientes a configurarem a solução de uma forma mais fácil, organizada, amigável e menos sujeita a erros. Estas ferramentas foram importantes para que a solução NGIN seja cada vez mais usada e da confiança dos clientes.

Quanto á utilização destas ferramentas a ferramenta *TestBeds* é usada internamente, por uma equipa de integração do *software* desenvolvido, já a ferramenta *Configuration Manager* é para uso dos clientes de forma a monitorar a solução NGIN, sendo assim a sua imagem nos clientes.

1.2. Contributos da tese

Como facilmente se deduz é necessário um infindável número de testes à solução NGIN, e especificamente ao sub-componente SLTF, para que esta, esteja suficientemente robusta no momento da

[1] O NGIN Care ou CARE é o subsistema de gestão de clientes, com funções de *Customer Care* e de aprovisionamento de clientes e serviços [1].

[2] O NGIN Core ou CORE é o subsistema que trata do controlo em tempo real da facturação dos clientes ao longo da chamada, sessão ou evento [1].

[3] Conjunto de módulos que compõem a solução NGIN, e que normalmente se enquadram no conjunto de conceitos de negócio que a solução possui [1].

instalação no cliente. Para que isto seja humanamente possível são necessários mecanismos de automatização de testes, e uma aplicação permitirá abranger um maior número de áreas de teste.

Assim surge a necessidade de desenvolver uma ferramenta capaz de efectuar esses testes, tendo como principais objectivos a:

- Capacidade de armazenar uma bateria de cenários de aprovisionamento, ou seja, ser possível através da ferramenta armazenar a informação e os dados correspondentes a uma versão de *software* igual à que será entregue ao cliente;
- Capacidade de armazenar uma bateria de cenários de configuração, ou seja, a capacidade de armazenar parâmetros e informações de configuração e variáveis genéricas reutilizáveis em diferentes testes;
- Capacidade de armazenar uma bateria de cenários de teste (combinações de cenários com resultados esperados). Este objectivo é muito importante pois esta bateria permitirá a execução dos testes de uma forma repetida e sempre que desejável;
- Produção e histórico dos resultados obtidos nos casos de teste;
- Desenvolvimento de uma interface gráfica apelativa para suportar a ferramenta, sendo muito importante ter uma interface intuitiva e fácil de utilizar, podendo assim explorar ao máximo os mecanismos de automatização de todo o processo.

A contribuição pessoal neste projecto passou por um envolvimento total em todas as fases de desenvolvimento do mesmo. Este envolvimento consuma-se nas fases de estudo e familiarização da linguagem de suporte à ferramenta, o PL/SQL (*Procedural Language/Structured Query Language*), estado da arte das ferramentas existentes nesta área, análise de requisitos, implementação, teste e documentação da solução.

Nos dias de hoje verifica-se uma grande agressividade do mercado, uma constante pressão dos clientes na obtenção de ferramentas em prazos cada vez mais apertados (importância elevada de um bom cenário de testes) e a necessidade dos próprios clientes poderem configurar os seus próprios serviços. As aplicações de configuração desempenham um importante papel nesta área, fornecendo aos clientes uma enorme vantagem, tornando as configurações mais amigáveis, potenciando a disponibilização de novos serviços em tempo cada vez mais rápido.

Em relação à configuração dos componentes NGIN, sempre que solicitado, são desenvolvidas ferramentas específicas à sua configuração. Os componentes da plataforma foram surgindo e a maneira de os configurar torna-se indispensável. Assim surgiu a necessidade de criar uma ferramenta capaz de permitir esta configuração, nasce assim a segunda ferramenta. Os objectivos desta ferramenta, o *Configuration Manager* incidem sobre:

- Coerência da configuração do NGIN CARE e do NGIN CORE, a ferramenta tem de configurar os dois componentes de uma forma clara e sempre que possível de uma forma a que o configurador não se aperceba se a informação que está a configurar vai ser usada pelo componente NGIN CARE, NGIN CORE ou em ambos;

- Abstracção de conceitos, sempre que possível alguns conceitos de negócio são "mascarados" para que a configuração seja feita baseada num contexto e de uma forma mais abstracta. A ferramenta tem de configurar os módulos que compõem a solução mas para o configurador é importante não ter a sensação que está a configurar os modelos de dados, diferenciando-se disso apenas o aspecto gráfico;
- Agilização da configuração com *wizards* condutores do processo, orientando a configuração por um conjunto de passos ordenados e organizados, sendo assim mais fácil construir uma configuração válida, correcta e completa.
- Herança de configurações e edição massiva de informação. Processos de simplificação e automatização são sempre importantes. A possibilidade de reutilização de configurações completas ou partes delas permite uma flexibilidade enorme ao configurador. Aliado a este mecanismo a possibilidade de editar informação comum massivamente permite uma enorme poupança de tempo;
- Navegação pelo mapa de interdependências, a ferramenta orienta o configurador pelas dependências que existem entre a configuração dos diferentes conceitos;
- Gestão da topologia e modelos activos. É possível através da ferramenta configurar não só conceitos de negócio mas também informação sobre plataformas e modelos de dados activos, onde as configurações vão ser transferidas para os modelos reais dos clientes, ou seja, através da aplicação é possível fazer a configuração da solução mas também transferi-la para os modelos activos das operadoras;
- Interfaces de consulta de informação de configuração por entidades externas, as configurações são possíveis de aceder via *Web Services* e não só apenas pelo CM;
- Mecanismos de extensão com outros sistemas ou ferramentas.

A contribuição pessoal neste projecto passou pela participação total nas áreas gráficas da aplicação. O envolvimento passou pelas fases de estudo e familiarização da linguagem de suporte à ferramenta, o GWT (*Google Web Toolkit*), implementação, teste e documentação desta parte da ferramenta. É importante dizer que a parte gráfica não se resume apenas ao desenho de ecrãs mas também, entre outras, ao tratamento da informação extraída da BD.

1.3.Estrutura da tese

O presente documento encontra-se estruturado em sete capítulos.

No capítulo 1, que agora se encerra, é descrito o contexto e a relevância deste trabalho, apresenta-se a motivação e os objectivos a atingir bem como uma breve descrição sobre as ferramentas desenvolvidas. É resumidamente apresentado o que foi elaborado durante este trabalho.

O capítulo 2, é apresentada uma breve descrição dos projectos desenvolvidos durante este período do mestrado, tendo como objectivo enquadrar o leitor para os restantes capítulos.

O capítulo 3 é dedicado ao estudo e síntese dos conceitos abordados e da área que a primeira ferramenta *TestBeds* se enquadra. É feita uma revisão do estado da arte, focando-se em temas como testes, automatização de teste, os diferentes tipos de testes.

O capítulo 4 é dedicado ao estudo e síntese dos conceitos abordados e da área que a ferramenta *Configuration Manager* se enquadra. É feita uma revisão do estado da arte, focando-se em temas como redes inteligentes e de próxima geração, e no conceito de catálogo de produtos e serviços.

No capítulo 5 é feita uma descrição detalhada da primeira ferramenta desenvolvida, *TestBeds*.

No capítulo 6 é feita a apresentação e a descrição de forma detalhada da segunda ferramenta desenvolvida, o *Configuration Manager*, neste capítulo é apresentada a arquitectura, a tecnologia e abordagens seguidas no seu desenvolvimento, bem como os principais aspectos técnicos implementados.

No capítulo 7, são apresentadas as principais conclusões, resultados e contribuições deste trabalho. Para terminar, são descritas algumas possibilidades de trabalho futuro.

Os anexos apresentam algumas partes da interface gráfica das ferramentas desenvolvidas.

2. Enquadramento do projecto

O objectivo deste segundo capítulo, é apresentar ao leitor uma breve descrição dos projectos desenvolvidos durante este trabalho de mestrado, tendo como objectivo primordial enquadrar o leitor para os restantes capítulos.

Antes de passar a uma apresentação das linguagens e tecnologias aplicadas nas ferramentas desenvolvidas, importa enquadrar e especificar as áreas de actuação destes. Dentro da solução NGIN foram desenvolvidas no âmbito deste projecto de mestrado duas ferramentas, a primeira ferramenta, *TestBeds*, a apresentar, na secção 2.2, refere-se a uma ferramenta de automatização de testes e validação de *software*, algo inovador para a empresa. A segunda ferramenta, *Configuration Manager*, prendeu-se com o desenvolvimento de uma aplicação para configuração de serviços e regras de megócios da plataforma NGIN (secção 2.3).

2.1.Descrição da solução NGIN

Abordando o conceito que suporta a solução NGIN, o conceito de rede inteligente, cientificamente pode ser visto como uma arquitectura de rede pública, adequada à criação e à exploração de serviços de telecomunicações num cenário multi-vendedor, destinando-se tanto para redes de telecomunicações fixas como móveis. A arquitectura reflecte a preocupação da separação entre o suporte lógico dos serviços e as funções de processamento e transporte das chamadas. A ideia base desta arquitectura consiste na existência de uma camada que se limita a efectuar o encaminhamento de chamadas e a gerir o tráfego, estando dotada de inteligência capaz de interpretar conjuntos de instruções dirigindo-as para os diversos serviços. Trata-se de uma camada extremamente flexível que admite com facilidade alterações nos serviços e é facilmente programável. A sua independência do hardware é obtida através da adopção do SS7[4] (sistema de sinalização normalizado número 7) [2]. A solução NGIN tem evoluído ao longo da última década de modo a satisfazer não só as necessidades do negócio nos mercados onde é utilizada, mas também a própria evolução tecnológica ao nível das redes e dos serviços. Entre as inúmeras características destacam-se as seguintes [1]:

- Flexibilidade na criação de serviços;
- Multi-tecnologia, Multi-operador (fixo, móvel), Multi-serviços;
- Disponibilidade de ferramentas de suporte ao negócio e gestão de rede;
- Alta disponibilidade;
- Escalabilidade;
- Arquitectura aberta (aderente a standards).

[4] O SS7 é baseado no modelo OSI e diferencia-se por existir um canal específico para troca de sinalização. Isto quer dizer que o canal de voz associado à chamada telefónica não é utilizado para troca de sinalização, mas sim um canal exclusivo para sinalização, comum a diversas chamadas [3].

A utilização da solução NGIN pelo Grupo PT começou com a implementação de serviços clássicos na rede fixa (ex: número verde, número azul, cartões de chamadas) e posteriormente nas redes móveis (serviços pré e pós pagos de voz), com a consequente adopção generalizada pelas várias empresas do Grupo. Numa segunda fase passaram também a ser abrangidas as redes de dados em pacotes (móveis e fixas) e o controlo de serviços *VoIP (Voice Over IP)*. A plataforma NGIN encontra-se neste momento completamente integrada na arquitectura IMS (*IP Multimedia Subsystem*). As principais vantagens na adopção da solução NGIN para os operadores são [1]:

- Disponibilização de uma solução integrada, desde a interacção da rede (SS7/Sigtran[5]/SIP[6]) até à execução de serviços, incluindo aprovisionamento da rede *online* e serviços de *customer care*;
- Controlo em tempo real da facturação dos clientes ao longo da chamada, sessão ou evento, o que elimina a possibilidade de fraude nos sistemas baseados em débitos executados após a conclusão da chamada, permitindo assim a convergência em tempo real da taxação e facturação;
- Previsão e controlo optimizado das receitas, e consequente aumento do *cash-flow*;
- Melhoria do *time-to-market* no desenvolvimento e disponibilização de novos serviços.

Paralelamente à evolução decorrente da maior abrangência nos cenários de utilização da plataforma, verificou-se uma crescente sofisticação dos serviços implementados, a capacidade de suportar, por exemplo, chamadas em espera e reencaminhamentos, pacotes em minutos, contas partilhadas, múltiplos saldos, bónus, descontos e promoções. A par disto a evolução dos serviços de dados, com *'Rating'* dependente de múltiplas variáveis, tais como, tempo, volume, conteúdos e localização. Não menos importante foi a implementação de planos mistos, como por exemplo empresariais e de empregados, pré-pago e pós-pago.

A sua componente de NGIN CARE disponibiliza funções de gestão de clientes, atendimento, provisionamento de serviços, gestão de promoções, de planos de preços e de recargas, etc.

A plataforma possui ainda um conjunto de ferramentas complementares, tais como [1]:

- *InoVox*, fornece aos operadores uma vasta gama de possibilidades para disponibilizar serviços de multimédia avançados;
- *NGIN Reporter*, que disponibiliza indicadores de desempenho dos serviços;
- *NGIN Mart*, que disponibiliza indicadores de desempenho do negócio;
- *NGIN Portal*, interface '*web'* para auto aprovisionamento dos clientes;
- *NGIN Manager*, que permite a gestão dos componentes da plataforma.

[5] SIGTRAN é o nome do grupo de trabalho da IETF *(Internet Engineering Task Force)* que tem desenvolvido uma série de protocolos que permitem transportar sinalização SS7 por redes IP. Por extensão chamamos SIGTRAN a este grupo de protocolos [4].

[6] *Session Initiation Protocol* – (SIP) é um protocolo de aplicação, que utiliza o modelo "*request-response*", similar ao HTTP, para iniciar sessões de comunicação interactiva entre utilizadores. É um padrão da IETF.

Hoje, a solução NGIN, é suportada numa arquitectura modular e estratificada permitindo uma evolução suave tanto a nível tecnológico como ao nível funcional seja no controlo de serviços em tempo real como na sua gestão integrada.

A figura 2.1 enquadra de um modo sintético todo a solução NGIN.

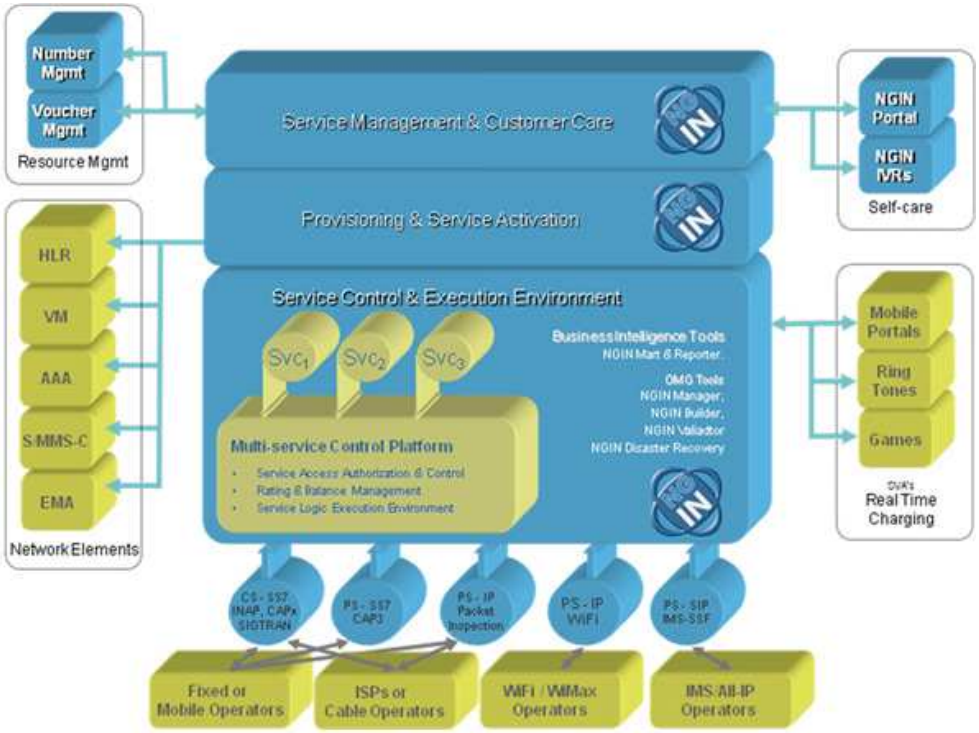

Figura 2.1 – Solução NGIN [1]

Basicamente, a arquitectura geral da solução NGIN, é composta por diversas camadas e por um conjunto adicional de funcionalidades autónomas, nomeadamente [1]:

- ***Service Management & Customer Care*** – onde é feita a gestão integral de clientes e serviços, com funções de *customer care*;
- ***Provisioning & Service Activation*** – camada responsável pela gestão do aprovisionamento das lógicas de serviço e dos diversos recursos de rede para activação/desactivação de serviços básicos;
- ***Service Control & Execution Environment*** – onde é feito o controlo da sinalização dos vários tipos de rede e o controlo em tempo real da execução das lógicas de serviço e de negócio;
- ***Business Intelligence Tools*** – composto por várias ferramentas de produção de relatórios e de gestão onde se destacam o *NGIN Manager*, *o NGIN Reporter* e o *NGIN Mart* descritos atrás;

Dentro da família NGIN, existem diversos produtos que se complementam. O produto INS enquadra-se na família de produtos NGIN, e na figura 2.2 é apresentado.

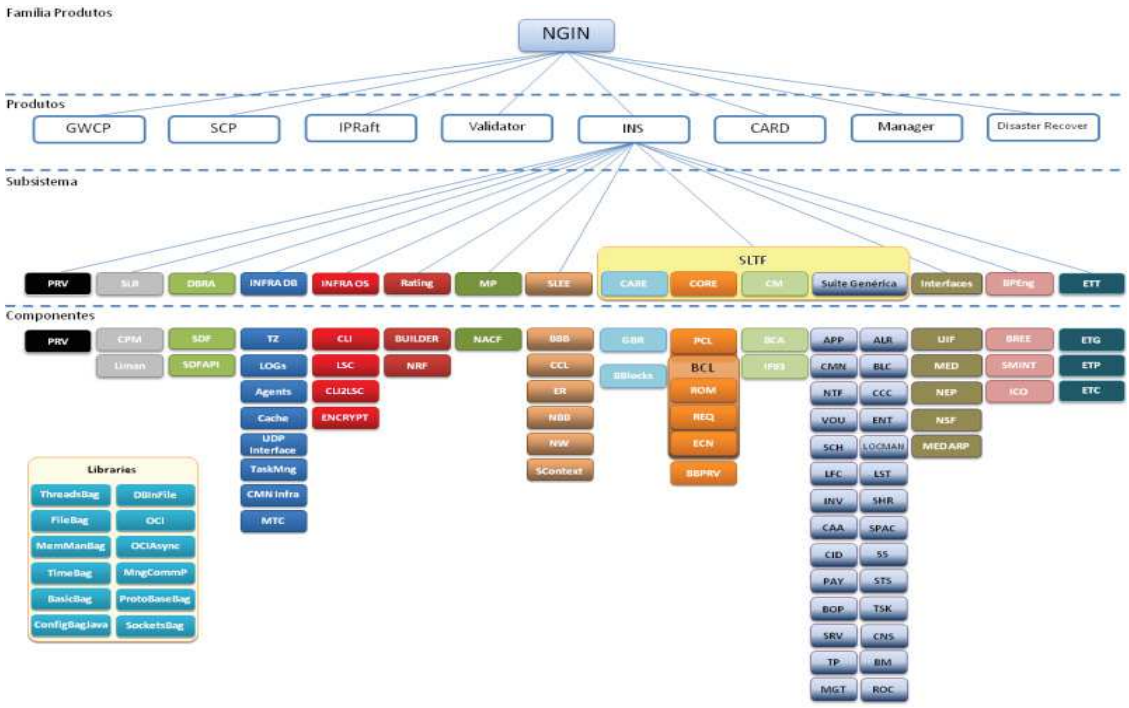

Figura 2.2 - Solução NGIN, focando o produto INS [1]

Usando novamente a figura 2.2 como ponto de análise é sobre os subsistemas NGIN CARE, NGIN CORE, CM e Suite genérica, denominado a este conjunto SLTF, que os projectos apresentados de seguida foram desenvolvidos.

Figura 2.3 – Subsistema SLTF [1]

A figura 2.3 apresenta o subsistema SLTF, e descodificando esse chavão e os seus componentes, temos o NGIN CARE como o subsistema de gestão de clientes da plataforma NGIN da PT Inovação, com funções de *Customer Care* e de aprovisionamento de clientes e serviços. É composta por duas camadas, uma orientada à gestão da aplicação de *call centrer* e outra orientada ao tratamento de eventos (por exemplo, recargas e outros pedidos de aprovisionamento).

O NGIN CORE é o subsistema que trata do controlo em tempo real da facturação dos clientes ao longo da chamada, sessão ou evento, o que elimina a possibilidade de fraude nos sistemas baseados em

débitos executados após a conclusão da chamada, permitindo assim a taxação e facturação convergente em tempo real. Consequentemente, existe uma optimização do controlo e previsão das receitas, aumentando o *cash-flow*, dotando assim a plataforma de um melhor tempo de resposta ao mercado no desenvolvimento e disponibilização de novos serviços.

A ferramenta de testes e validação de *software, TestBeds,* incide exclusivamente sobre os módulos que constituem a suite genérica, listadas na figura 2.3. A ferramenta actua sobre os procedimentos que constituem os módulos. O funcionamento passa pela invocação directa desses procedimentos, de forma a poder testar cada um deles e poder simular a sua utilização usando dados semelhantes aos reais.

A ferramenta de configuração, *CM*, permite a definição de regras de negócio abstraindo detalhes de funcionamento dos sistemas configurados para conceitos de negócio, ou seja, a ferramenta permite configurar de uma forma a que o utilizador não tenha essa percepção do modelo de dados dos módulos da suite genérica, e dos sistemas NGIN CARE e NGIN CORE.

Resumindo ambas as ferramentas têm em comum algo, o uso do subsistema SLTF materializada no uso do modelo de dados da suite genérica. A primeira, permitindo testar o funcionamento interno dos módulos que compõem a suite, a segunda permitindo configurar os modelos de dados que derivam desses módulos e que permitem que o negócio e as suas regras tenham o funcionamento esperado pelos clientes da solução NGIN.

2.2.Projecto TestBeds

Como facilmente se deduz, é necessário um infindável número de testes ao sub-componente, suite genérica, que é composto por diversos módulos (desenvolvidos em PL/SQL) para que, quando chegue ao cliente não comprometa o funcionamento de toda a solução. Para que isto seja humanamente possível são necessários mecanismos de automatização de testes, e uma ferramenta desenvolvida tendo em conta as necessidades reais permite abranger um maior número de áreas de teste.

A tarefa de efectuar testes de *software* foi considerada secundária durante muito tempo. Geralmente era vista com um "castigo" para o programador, ou como, uma tarefa onde não se deveria gastar muito tempo nem dinheiro. O tema esteve relegado para segundo plano, e, até a alguns anos atrás não se encontrava muita literatura sobre o assunto.

"É do conhecimento de todos entre os profissionais de software que nunca se elimina o último 'bug' de um programa. Os 'bugs' são aceites como uma triste realidade. Espera-se eliminá-los a todos, um a um, mas sabe-se que nunca deixarão de existir.", afirmou *DeMarco* [6].

Esta problemática de testes, correcção de *bugs*, validações de código, levanta inúmeras interrogações. "O que são testes de *software*?", para *Myers* [7] , *"O teste de 'software' consiste em executar um programa com a intenção de encontrar 'bugs'"*, já a *Burnstein* [8] define como *"um grupo de casos de teste relacionados, ou um grupo de casos de teste e procedimentos relacionados"*. Dentro da problemática de testes de *software* existem diferentes tipos, muitas vezes interligados e normalmente usados num mesmo processo de testes, de forma a abranger diferentes áreas.

Executar autonomamente estes testes muitas vezes torna-se humanamente impossível, daí ser necessário usar na maioria dos casos formas de automatização de testes, e nada melhor que uma ferramenta de testes que facilite este processo. Automatização de testes pode ser definida pelo *"uso de 'software' para a controlar a execução de testes, a comparação dos resultados esperados com os resultados obtidos, a configuração de pré-condições de teste e outras funções de controlo e relatório de teste."*.

Nos dias de hoje já existem um número significativo de ferramentas de teste, umas mais específicas para um tipo de testes, outras com uma capacidade mais global que conseguem abranger diferentes tipos. Mas com essas ferramentas, muitas delas até '*open-source'* surgem algumas interrogações. *"É necessário construir uma ferramenta de raiz?","Será que as ferramentas que existem no mercado respondem a todas as necessidades?", "Será que fazem o que é esperado?", "Será que não têm funcionalidades a mais do que o necessário?", "Uma ferramenta feita à medida das necessidades ou uma ferramenta 'comercial'?"*

Todas estas questões foram colocadas inicialmente, foram debatidas e continham prós e contras. Após o debate de ideias a opção caiu sobre a criação de uma ferramenta com o apoio da equipa que preferencialmente a vai usar, definindo bem aos requisitos e necessidades desta. O grande factor de decisão foi sem dúvida a de desenvolver uma ferramenta proprietária que fosse capaz de testar uma parte do *software* desenvolvido na PT Inovação. A ferramenta desenvolvida serve para responder aos requisitos especificados e apresentados no capítulo 5, onde o que se deseja é que a mesma faça nem mais nem menos do que se espera. Outras razões também serviram para esta escolha, do estudo feito, não existiam ferramentas capazes de se adaptarem facilmente ao modelo de dados existente nem que fossem capazes de tratar os tipos de dados proprietários da solução NGIN.

A ferramenta *TestBeds* surge da necessidade de criar uma plataforma capaz de automatizar testes e validações de *software*. Com o elevado ritmo de construção, evolução e manutenção de *software*, surge permanentemente a necessidade de centralizar o controlo de testes e de avaliação de desempenho para que as entregas ao cliente sejam efectuadas com o mínimo de falhas e com a máxima robustez. Assim, surgiu a necessidade de criar uma plataforma que fosse capaz de automatizar e tratar os testes de forma a avaliar a desempenho dos módulos desenvolvidos, mas também a evolução do seu funcionamento face a versões anteriormente testadas. Esta fase de testes de *software* é um dos sub-processos mais propensos a serem automatizados, devido à sua natureza repetitiva e à necessidade de cadastro, produção de relatórios e análise de resultados, sendo por isso fulcral uma ferramenta que automatize ao máximo este processo.

A ferramenta de testes desenvolvida, *TestBeds*, vai actuar exclusivamente sobre a suite genérica, sobre os módulos que a compõem. Estes pacotes de *software* são desenvolvidos em PL/SQL e a ferramenta de testes tem de ter a capacidade de descobrir a composição destes procedimentos, ou seja, a primeira função da ferramenta passa pela capacidade desta descodificar essas funções, os seus parâmetros, e dotar-se de uma inteligência que a capacite interpretar estes dados. Estes pacotes que internamente correspondem a módulos em PL/SQL, são desenvolvidos sobre uma Base de Dados em *Oracle*, e a primeira etapa da ferramenta passa por percorrer o dicionário de dados da base de dados onde estão definidos os pacotes referentes aos módulos de forma a codificar estes procedimentos em objectos

conhecidos e que facilmente sejam interpretados pela ferramenta. Esta parte foi desenvolvida também recorrendo à linguagem PL/SQL, e o que faz é mediante uma série de *queries* em *SQL* descobrir a assinatura dos procedimentos internos definidos em cada pacote e transformá-los em objectos conhecidos na aplicação. A segunda etapa será configurar estes objectos definindo valores de entrada e valores de saída esperados para os procedimentos que se pretende executar. Por fim os testes podem ser executados e são avaliados com valores OK/NOK.

Estas duas etapas têm um suporte de uma interface gráfica desenvolvida em GWT, permitindo assim ao utilizador poder manipular de uma forma mais amigável a ferramenta. Depois de avaliados os procedimentos que constituem o plano de testes, se os resultados forem bem sucedidos, a ferramenta valida que os módulos desenvolvidos, estando portanto, prontos a serem usados.

2.3.Projecto Configuration Manager

Um produto complexo como é a solução NGIN, necessita de uma ferramenta que apoie o seu manuseamento e a sua configuração. Como foi referido anteriormente o subsistema SLTF é composto pelos componentes NGIN Care, NGIN Core, CM e a suite genérica, mas nem sempre assim foi.

Até 2008, o subsistema SLTF era composto pelo CORE, suite genérica e PCA (*Package Configuration Application*). A ferramenta PCA foi a primeira ferramenta de configuração da plataforma NGIN, que trabalha directamente sobre o modelo de dados passivo dos pacotes genéricos. Esta ferramenta permite a configuração apenas de produtos CORE.

Uma necessidade que se tornou indispensável face ao número de clientes que dispunham da solução, foi produtizar o conceito de NGIN Care. Até aqui o CARE não fazia parte do conceito do subsistema SLTF, era cada equipa de customização de cada cliente que o desenvolvia, (existia dentro das equipas de desenvolvimento pessoas que só programavam para um determinado cliente), no entanto face ao esforço, face à duplicação de código, face a muitos dos desenvolvimentos serem semelhantes e desenvolvidos com o mesmo propósito, foi decidido incluir um conceito de CARE no SLTF. Passou-se então de um produto só CORE, para um produto CARE e CORE. Surge desde logo um problema. A ferramenta, PCA, que existia apenas configura CORE, se o produto agora é CARE e CORE deixou de haver uma aplicação que fosse capaz de o configurar.

Com este novo cenário algumas questões são levantadas. "*Evoluir a ferramenta PCA para que esta dê suporte a esta nova realidade ou criar uma ferramenta de raiz?"*. A opção recaiu sobre uma nova ferramenta, uma ferramenta mais apelativa visualmente ao utilizador, que abstraí-se ao máximo de conceitos de negócio, e que cumpra todos os objectivos posteriormente apresentados e detalhados no capítulo 6. Surge então o *Configuration Manager* (CM), que consiste numa aplicação composta por um conjunto de módulos que cooperam entre si, com o objectivo de permitir a definição de entidades e regras de negócio que determinam o comportamento do sistema NGIN.. Assim o objectivo foi passar de uma ferramenta pouco amigável e que configurava parte do sistema NGIN para uma ferramenta que orientasse o utilizador nas configurações, tanto de CARE como de CORE, tendo como principais funcionalidades: um mecanismo de navegação pelo mapa de dependências entre configurações; edição massiva de configurações; gestão de topologia e modelos activos.

2.4. Ferramentas utilizadas

Este subcapítulo tem como objectivo apresentar as principais ferramentas/técnicas/linguagens usadas no desenvolvimento dos projectos.

O PL/SQL (*Procedural Language/Structured Query Language*) é uma extensão da linguagem padrão SQL. O PL/SQL pode ser usado em bases de dados *Oracle*, num Servidor *Oracle*, em ferramentas clientes, isto é, *Oracle Forms*. O PL/SQL é muito similar à linguagem SQL mas acrescenta construções de programação similares a outras linguagens. Permite que a manipulação de dados seja incluída em unidades de programas. Blocos de PL/SQL são passados e processados pelo PL/SQL *Engine* que pode estar inserido numa ferramenta *Oracle* ou do servidor. O PL/SQL *Engine* filtra os comandos SQL e encaminha individualmente o comando SQL para o SQL *Statement*, que processa o PL/SQL com os dados retornados do servidor. É a linguagem básica para criar programas complexos e poderosos, não só em bases de dados, mas também em diversas ferramentas *Oracle*. Esta linguagem foi essencial no desenvolvimento da ferramenta *TestBeds*, pois foi sobre esta que todo o funcionamento interno foi feito, isto porque, a ferramenta vai actuar sobre módulos desenvolvidos nesta linguagem, logo só seria possível desenvolver a aplicação nesta linguagem.

XML *(eXtensible Markup Language)* é uma recomendação da W3C[7] para gerar linguagens de marcação para necessidades especiais. O seu objectivo principal é a facilidade de partilha de informação. A sua filosofia incorpora vários princípios importantes:

- Separação do conteúdo da formatação;
- Simplicidade e legibilidade, tanto para humanos como para computadores;
- Possibilidade de criação de *tags* ilimitadas;
- Concentração na estrutura da informação, e não na sua aparência.

O XML é considerado um bom formato para a criação de documentos com dados organizados de forma hierárquica. Pela sua portabilidade, é fácil a uma aplicação escrever num ficheiro XML, e uma outra aplicação distinta ler estes mesmos dados e interpretá-los da mesma forma [9].

Esta foi a abordagem seguida na implementação da ferramenta *TestBeds*, a aplicação tem a capacidade de criar funções dos tipos não nativos da *Oracle*, codificá-los numa estrutura em XML para que a aplicação interprete estes objectos. A utilização deste formato permite facilmente codificar os tipos nativos nestas funções em XML, e o seu inverso. Os termos Serialização (codificação numa estrutura XML), e Deserialização (descodificação para o formato original), foram criados para definir estes processos.

A linguagem Java surge nestes projectos essencialmente para dar suporte ao interface gráfico. O Java é uma linguagem de programação orientada a objectos desenvolvida na década de 90 por uma equipa de programadores dirigida por *James Gosling*, na *Sun Microsystems*. Uma das grandes diferenças para outras linguagens convencionais, que são compiladas para o código nativo, é que a linguagem Java é

[7] *World Wide Web Consortium* (W3C), é uma organização internacional, que agrega empresas, órgãos governamentais e organizações independentes, e que visa desenvolver padrões para a criação e a interpretação de conteúdos para a Web [11].

compilada para um '*bytecode'* que é executado por uma máquina virtual. A linguagem foi projectada tendo em vista os seguintes objectivos [10]:

- Orientação a objectos;
- Portabilidade - Independência de plataforma;
- Recursos de rede – Possui uma extensa biblioteca de rotinas que facilitam a cooperação com os principais protocolos;
- Segurança - Pode executar programas via rede com restrições de execução.

Além destes aspectos, podem-se ainda destacar outras vantagens apresentadas pela linguagem:

- Sintaxe similar a Linguagem C/C++;
- Facilidades de internacionalização;
- Simplicidade na especificação, tanto da linguagem como do "ambiente" de execução (JVM);
- É distribuída com um vasto conjunto de bibliotecas (ou APIs);
- Possui facilidades para criação de programas distribuídos e multitarefa;
- Desalocação de memória automática (*garbage collector*).

Aliado à linguagem anteriormente apresentada, o GWT é descrito como um *framework open source*[8] que permite o desenvolvimento de aplicações web de uma forma fácil a quem tem conhecimentos de programação em Java, sem que para isso seja necessário qualquer conhecimento de *Javascript*[9], HTML[10] (*Hypertext Transfer Protocol*) e CSS[11] (*Cascading Style Sheet*). Toda a aplicação é desenvolvida em Java, sendo que uma aplicação GWT está dividida em duas partes, a parte cliente e a parte servidor. O que esta *framework* faz é compilar todo o código desenvolvido em Java na parte cliente para *Javascript*, permitindo, desta forma, que a aplicação seja executada em qualquer *browser*, uma vez que também é garantido pela *Google* que todas as aplicações desenvolvidas em GWT são compatíveis com qualquer *browser* dos mais utilizados hoje em dia (Internet Explorer, Mozilla Firefox, Safari, Opera, Chrome) [12].

Quando se está a desenvolver esta parte da aplicação há que ter em atenção que irá ser compilada para *Javascript*, sendo que, nem tudo é suportado pois o compilador de Java-para-Javascript apenas disponibiliza um emulador da biblioteca JRE (*Java Runtime Environment)*, ou seja, só uma parte das classes J2SE *(Java 2 Platform, Standard Edition)* e J2EE *(Java 2 Enterprise Edition)* é que são

[8] *Software* de utilização livre, em que todos podem contribuir, seja no seu desenvolvimento, na correcção de erros, na documentação, desde que esta condição de software livre liberdade seja mantida [13].

[9] É uma linguagem de programação criada para atender, principalmente, a validações de formulários no lado cliente e interacção com as páginas. *JavaScript* tem sintaxe semelhante à do Java, mas é totalmente diferente no conceito e no uso [13].

[10] É uma linguagem de marcação utilizada para produzir páginas na Web. Documentos HTML podem ser interpretados por *browsers* [15].

[11] É uma linguagem de estilos utilizada para definir a apresentação de documentos escritos numa linguagem de marcação, como HTML ou XML. O principal benefício é separar o formato e o conteúdo de um documento [16].

suportadas. Ao contrário do que acontece na parte cliente da aplicação, na parte servidor não existe qualquer limitação no uso de bibliotecas *Java*, sendo permitido o uso de qualquer tecnologia. Como uma aplicação "normal", às vezes é necessário que a aplicação comunique com o servidor, por exemplo, acessos a base de dados para extrair ou para guardar dados. Para isto, o GWT dispõe de um mecanismo de comunicação entre cliente e servidor denominado de GWT-RPC (*Google Web Toolkit - Remote Procedure Call*), bastando para isso que a parte cliente faça um pedido ao servidor (através do *browser*). Tudo isto é feito de forma assíncrona, não ficando o cliente bloqueado à espera da resposta do pedido feito. Como se pode ver na figura 2.5, a criação de um serviço traduz-se na criação de duas interfaces, uma síncrona e outra assíncrona, e a criação de uma *servlet*, no lado do servidor. As duas interfaces servem para definir as operações que o serviço irá conter, por exemplo, as operações de guardar dados nas bases de dados. O *servlet* contém a implementação dessas operações, implementando a interface síncrona.

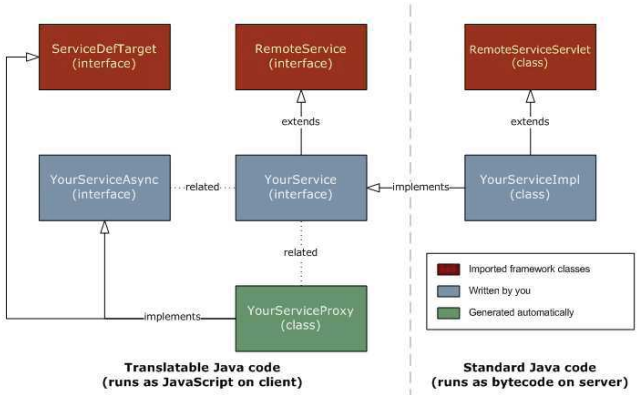

Figura 2.4 - Implementação de um serviço RPC [12]

Esta é das partes mais importantes de uma aplicação GWT. Todos os pedidos feitos ao servidor são assíncronos, sendo necessário definir um método de *callback* para tratar a resposta do servidor quando esta chegar ao cliente. A razão para todas as chamadas ao servidor serem assíncronas é simples, os motores de *Javascript* dos *browsers* são *single-threaded* o que se traduziria num bloqueio da aplicação a cada chamada ao método *XMLHTTPRequest* (objecto *Javascript* definido pelos *browsers* para enviar pedidos HTTP), aquando de qualquer pedido do cliente ao servidor [12]. O GWT foi a ferramenta usada para o desenvolvimento das interfaces gráficas dos dois projectos.

ETL (*Extract Transform Load)*, é um processo de *software* cuja função é a extracção de dados de diversos sistemas, transformação desses dados conforme regras de negócio e por fim o armazenamento dos dados num *data mart* ou *data warehouse*. Os projectos de *data warehouse* consolidam dados de diferentes fontes, a maioria dessas fontes tendem a ser bases de dados relacionais ou *flat files*, mas podem existir outras fontes. Um sistema ETL tem que ser capaz de comunicar com as bases de dados e ler diversos formatos de ficheiros utilizados por toda a organização. Essa pode ser uma tarefa pouco trivial, e muitas fontes de dados podem não ser acedidas muito facilmente [17]. Uma aplicação muito usada deste tipo, é o *Kettle Pentaho Data Integration.* A ferramenta *Configuration Manager* usa esta ferramenta para migrar dados entre bases de dados.

Castor – Source Generator é uma ferramenta *open source* em *Java* para *databinding*. É uma forma de relacionar os objectos *Java*, documentos XML e tabelas relacionais [18].

O projecto iBATIS é uma ferramenta de mapeamento entre objectos (*Java, .NET e Ruby*) e modelos relacionais. O iBATIS é usado principalmente para *data access object* (DAO) e *object-relational mapping* (ORM). A maior vantagem deste programa é a sua simplicidade em relação a outras ferramentas do género [19].

3. A importância dos testes de *software*

Os testes de *software* têm cada vez mais importância nas organizações. Este capítulo encontra-se organizado em quatro secções. Na primeira secção são descritos as principais abordagens e perspectivas de diferentes autores sobre testes de *software*. Na segunda secção é apresentada uma análise sobre testes e qualidade de *software.* Na terceira secção é explorado o tema de automatização de testes. Para terminar, são apresentadas as principais conclusões resultantes da análise da problemática de testes de *software,* e a classificação adoptada no âmbito da ferramenta *TestBeds.*

3.1. Testes de software - Conceitos base

A engenharia de *software* evoluiu significativamente nas últimas décadas pretendendo estabelecer técnicas, critérios, métodos e ferramentas para a produção de *software*, e o aumento da utilização de sistemas informáticos em praticamente todas as áreas da actividade humana, provocou uma crescente procura de qualidade e produtividade, tanto do ponto de vista do processo de produção como do ponto de vista dos produtos gerados. O processo de desenvolvimento de *software* envolve uma série de actividades nas quais, apesar das técnicas, métodos e ferramentas usadas, podem ocorrer erros no produto desenvolvido. Actividades agregadas sob a procura de garantia de qualidade de *software* têm sido introduzidas ao longo de todo o processo de desenvolvimento, entre elas as actividades de VV&T (Verificação, Validação e Teste), com o objectivo de minimizar a ocorrência de erros e riscos associados [20]. Entre as técnicas de verificação e validação, a actividade de teste é uma das mais utilizadas, tornando-se como um dos elementos para fornecer confiança ao *software* em complemento a outras actividades, como por exemplo, o uso de revisões e de técnicas formais e rigorosas de especificação e de verificação [21].

A actividade de verificação, diz respeito ao conjunto de actividades que garantem que o *software* implementa correctamente uma função específica. O objectivo do processo de verificação é avaliar a consistência de uma implementação de acordo com uma especificação, assegurando que o produto será completo e correcto.

A actividade de validação, é o processo de avaliar a qualidade do *software*, validando, se o mesmo está a desempenhar o que foi especificado, no sentido de atender às reais necessidades do utilizador. Tem como objectivo revelar as falhas de especificação e erros de codificação, assegurando que o produto final corresponda ao que foi solicitado.

A actividade de teste consiste numa análise dinâmica do produto de *software* e é uma actividade relevante para a identificação e eliminação de erros que existam. Salienta-se que a actividade de teste tem sido apontada como uma das mais custosas no desenvolvimento de *software*. Apesar deste facto, *Myers* relata que aparentemente o conhecimento sobre testes de *software* é muito menor que sobre outras actividades do desenvolvimento de *software* [7].

3.1.1. Defeito, Engano, Erro e Falha

Antes de apresentar o conceito de teste é importante definir os conceitos de defeito, engano, erro e falha. A IEEE (*Institute of Electrical and Electronics Engineers*) tem realizado vários esforços de padronização, entre eles padronizar a terminologia utilizada no contexto de engenharia de *software*. O padrão IEEE 610.12-1990 diferencia os termos: defeito, engano, erro e falha. Assim, um defeito (*fault*) é um acto inconsistente cometido por um indivíduo ao tentar entender uma determinada informação, resolver um problema ou utilizar um método ou uma ferramenta. Por exemplo, uma instrução ou comando incorrecto; um engano (*mistake*) é acção humana que produz um resultado incorrecto, como por exemplo, uma acção incorrecta tomada pelo programador; um erro (*error*) é uma ideia errada/equívoco, ou interpretação errada/má compreensão por parte do programador, é uma manifestação concreta de um defeito num artefacto de *software*. São normalmente denominados como *"bugs"*. A diferença entre o resultado obtido e o resultado esperado, ou seja, qualquer estado intermediário incorrecto ou resultado inesperado na execução de um programa constitui um erro; e falha (*failure*) é o comportamento operacional do *software* diferente do esperado pelo utilizador. Uma falha pode ter como causa a ocorrência de diversos erros e alguns erros podem nunca causar uma falha, os termos engano, defeito e erro são referenciados como erro (causa) e o termo falha (consequência) como um comportamento incorrecto do programa [22].

De uma forma geral, os erros são classificados em, erros computacionais onde o erro provoca uma computação incorrecta mas o caminho executado (sequências de comandos) é igual ao caminho esperado; e erros de domínio em que o caminho efectivamente executado é diferente do caminho esperado, ou seja, um caminho errado é seleccionado.

A figura 3.1 expressa a diferença entre estes conceitos. Defeitos fazem parte do universo físico (a aplicação propriamente dita) e são causados por pessoas, por exemplo, através do mau uso de uma tecnologia. Defeitos podem ocasionar a manifestação de erros num produto, ou seja, a construção de um *software* de forma diferente do que foi especificado (universo de informação). Por fim, os erros originam falhas, que são comportamentos inesperados do *software* que afectam directamente o utilizador final da aplicação (universo do utilizador) e pode inviabilizar a utilização do *software* [22].

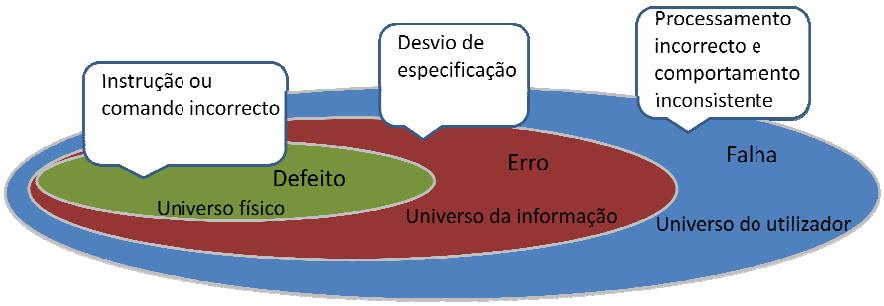

Figura 3.1 – Defeito X Erro X Falha [22]

3.1.2. O Processo de teste de *software*

O teste de produtos de *software* envolve basicamente quatro etapas: planeamento de testes, desenho dos casos de teste, execução e avaliação dos resultados dos testes [7]. Estas actividades devem ser desenvolvidas ao longo do próprio processo de desenvolvimento de *software*, e em geral, concretizam-se em quatro fases de teste: de unidade, de integração, de sistema e de aceitação, como apresentado na figura 3.2.

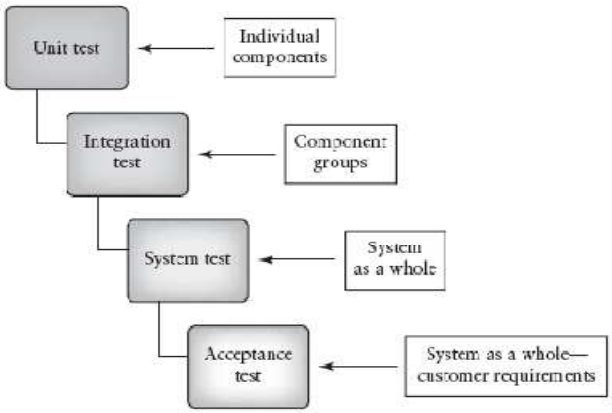

Figura 3.2 – Níveis ou fases de testes [8]

O teste unitário concentra esforços na menor unidade do projecto de *software*, ou seja, procura identificar separadamente erros de lógica e de implementação em cada módulo do *software*. Nos testes unitários, os diversos componentes aplicacionais são codificados e testados de forma isolada, garantindo assim a respectiva correcção interna. Incidem sobre partes do sistema, e são realizados por cada programador de forma independente. Este nível de teste é baseado na experiência, especificações e código, sendo o seu principal objectivo detectar defeitos funcionais e estruturais em partes de código.

O teste de integração é uma actividade sistemática aplicada durante a integração da estrutura do programa visando a descoberta de erros associados às interfaces entre os módulos. O objectivo é, a partir dos módulos testados ao nível da unidade, construir a estrutura do programa que foi determinada pelo projecto. Os testes de integração, são testes segmentados, que vários programadores realizam em conjunto com vista a garantir que vários componentes interagem entre si de forma adequada.

O teste de sistema, realizado após a integração do sistema, visa identificar erros de funções e características de desempenho que não estejam de acordo com a especificação. Os testes de sistema, são testes globais em que todos os componentes do sistema são integrados. Possibilitam a verificação da conformidade do sistema com todos os requisitos definidos, normalmente são da responsabilidade de uma equipa de testes independente. Este nível de teste é normalmente baseado num documento de requisitos (requisitos/especificações funcionais e requisitos de qualidade), sendo o seu principal objectivo o de avaliar atributos tais como usabilidade, fiabilidade e desempenho (assumindo que os testes unitários e de integração foram realizados) [20].

Os testes de aceitação, são testes formais que os utilizadores realizam sobre o sistema. Quando o sistema passa este "difícil" teste, o cliente deverá aceitar o sistema como "pronto" e consequentemente

este pode ser colocado em produção, ou operação, efectiva. O seu principal objectivo é avaliar se o produto vai de encontro aos requisitos e expectativa do cliente [8] [20].

A figura 3.3 espelha a relação entre os níveis ou fases dos testes, as estratégias ou abordagens de desenho de casos de teste e os atributos de qualidade.

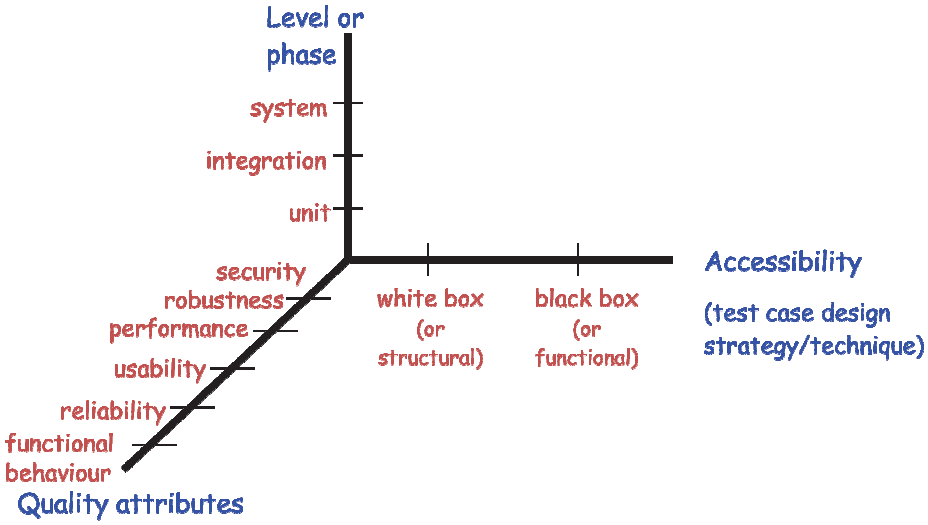

Figura 3.3 - Níveis ou fases dos testes, as estratégias ou abordagens de desenho de casos de teste e os atributos de qualidade [8]

Um ponto crucial na actividade dos testes, independentemente da fase, é o desenho e/ou a avaliação da qualidade de um determinado conjunto de casos de testes utilizados para o teste do produto de *software*, dado que em geral, é impraticável utilizar todo o domínio de *inputs* de dados para avaliar os aspectos funcionais e operacionais de um produto de *software* em teste. O objectivo é utilizar casos de teste que tenham alta probabilidade de encontrar a maioria dos defeitos com um mínimo de tempo e esforço. Segundo *Myers*, o principal objectivo do teste de *software* é *"revelar a presença de erros no produto"*. Portanto, um teste bem sucedido é aquele que consegue determinar casos de teste para os quais o programa em teste falhe [7].

O planeamento dos testes deve ocorrer em diferentes níveis e em paralelo com o desenvolvimento do *software*, dessa forma, observando a figura 3.4, o planeamento e o projecto dos testes devem ocorrer de cima para baixo, ou seja, inicialmente é planeado o teste de aceitação a partir do documento de requisitos, seguindo-se o planeamento do teste de sistema a partir do projecto de alto nível do *software*, em seguida ocorre o planeamento dos testes de integração a partir do projecto detalhado, e por fim, o planeamento dos testes a partir da codificação. Já a execução ocorre no sentido inverso. [23]

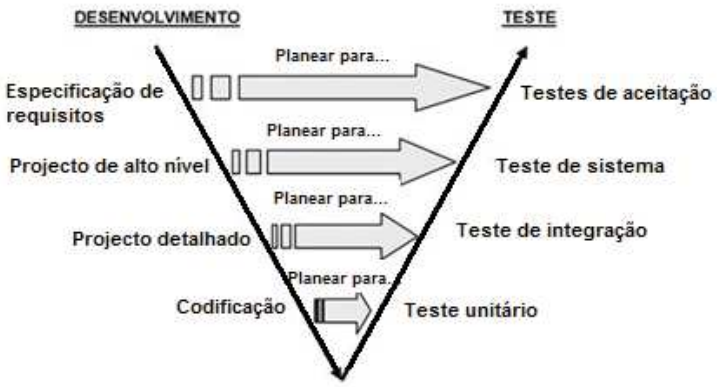

Figura 3.4 - Modelo em V que descreve o paralelismo entre as actividades de desenvolvimento e teste de *software* [23]

Voltando à figura 3.3 existe também forma de classificar os testes quanto aos seus atributos de qualidade. Os testes de desempenho permitem analisar o tempo de resposta do sistema e, de um modo geral, verificar o nível de utilização de recursos disponíveis. Os testes de usabilidade permitem analisar a adequabilidade do desenho das interfaces homem-máquina e validar, se o sistema é fácil de utilizar. Os testes funcionais têm como objectivo determinar a correcção da implementação de funcionalidades, conforme especificadas pelos correspondentes requisitos funcionais. Nos testes de segurança procuram-se os comportamentos anómalos que não se sabem quando acontecem. Precaver contra ataques, garantir a robustez do *software* face a determinados ataques típicos, detectar vulnerabilidades, preparar medidas de contingência, são as principais preocupações deste tipo de testes. Nos testes de robustez o objectivo é determinar o comportamento de um sistema em situações hostis e normalmente são pensados em conjunto com os testes de segurança. Os testes de fiabilidade têm como objectivo avaliar a capacidade de um sistema de *software* desempenhar as suas funções sob determinadas condições num determinado período de tempo [8] [20].

3.1.3. Abordagens de desenho de casos de teste

Em geral, os critérios de teste de *software* são estabelecidos, basicamente, a partir de três abordagens: a funcional, a estrutural e a baseada em erros.

No teste baseado em especificação ou funcional ou como também é conhecido teste caixa-preta pelo facto de tratar o *software* como uma caixa cujo conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os *inputs* de dados e os resultados obtidos. Na técnica de teste funcional são verificadas as funções do sistema sem existir preocupação com os detalhes de implementação, é avaliado o comportamento externo do produto de *software*, sem se considerar o comportamento interno do mesmo. Apenas são consideradas as entradas e saídas como uma base para o desenho dos casos de teste [24].

Os critérios e requisitos de teste são estabelecidos a partir da especificação do *software*. A grande função deste tipo de testes é determinar se o programa satisfaz os requisitos funcionais e não-funcionais que foram especificados, ou seja, valida se o que foi especificado foi implementado correctamente. É executado considerando como base os requisitos e as funcionalidades do *software*, quanto mais entradas

de dados são fornecidas, mais rico será o teste. Numa situação ideal todas as entradas de dados possíveis seriam testadas, mas na ampla maioria dos casos isso é impossível [8] [24]. O problema é que, em geral, a especificação existente é informal e, desse modo, a determinação da cobertura total da especificação que foi obtida por um dado conjunto de casos de teste também é ela informal [25]. Entretanto, os critérios de teste baseados em especificação podem ser utilizados em qualquer contexto (procedimental ou orientado a objectos) e em qualquer fase de teste sem a necessidade de modificação. Alguns métodos possíveis para estes tipos de testes funcionais são [20]:

1) ***Equivalence class partitioning*** **(Partição em classes de equivalência) -** A partir dos *inputs* de dados identificados na especificação, divide-se o domínio de entrada do programa em classes de equivalência válidas e inválidas. Em seguida selecciona-se o menor número possível de casos de teste, baseando-se na hipótese de que um elemento de uma dada classe seria representativo da classe toda, sendo que para cada uma das classes inválidas deve ser gerado um caso de teste distinto. Para cada operação, o "tester[12]" deve identificar as classes de equivalência dos argumentos e os estados dos objectos. O uso de particionamento permite examinar os requisitos mais sistematicamente e restringir o número de casos de teste existentes. Alguns autores também consideram o domínio de saída do programa para estabelecer as classes de equivalência.

2) ***Boundary Value Analysis*** **(Análise de valor limite) -** É um complemento ao critério de partição em classes de equivalência, sendo que os limites associados aos *inputs* são exercitados de forma mais rigorosa; ao invés de se seleccionar um elemento qualquer de uma classe, os casos de teste são escolhidos nas fronteiras das classes (problemas com índices de *arrays*, decisões, *overflow*, etc), pois nesses pontos concentra-se um grande número de erros. Se o sistema se comportar bem nos casos fronteira então provavelmente comportar-se-á bem em valores intermédios. O espaço de saída do programa também é particionado e são exigidos casos de teste que produzam resultados nos limites dessas classes de saída.

3) **Cause-and-Effect Graphing (Técnicas de grafos de causa-efeito) -** Os critérios anteriores não exploram combinações das condições de entrada. Este critério estabelece requisitos de teste baseados nas possíveis combinações das condições de *inputs* de dados. Primeiramente, são levantadas as possíveis condições de entrada (causas) e as possíveis acções (efeitos) do programa. A seguir é construído um grafo que relaciona as causas e efeitos analisados. Esse grafo é convertido numa tabela de decisão a partir da qual são derivados os casos de teste.

4) **Random testing -** Nesta técnica os valores são gerados aleatoriamente, o que por si só levanta um elevado número de interrogações. "Serão os *inputs* suficientes?", "Se fossem outros os resultados seriam melhores?", "Será que os valores escolhidos conseguem evidenciar uma percentagem aceitável de defeitos que possam existir?". De facto este tipo de testes poupam tempo e esforço mas revelam pouca eficiência.

[12] É o responsável por testar o produto de *software*. Realiza especificamente as tarefas que foram designadas pelo gestor do projecto. O *tester* deve ter a responsabilidade de executar os casos de teste e criar *scripts* para verificar se os requisitos estão de acordo com o que o cliente solicitou

A estratégia de e*quivalence class partitioning* é uma ferramenta muito útil no domínio da especificação de testes de caixa-preta. Contudo o *tester* terá de ter conhecimento da especificação dos *inputs/outputs* e os respectivos comportamentos do sistema. Esta estratégia torna-se muito mais potente quando usada em conjunto com a estratégia de *boundary value analysis*, pois normalmente os *bugs* ocorrem abaixo, acima e nos limites das classes de equivalência. No entanto, através da estratégia de *equivalence class partitioning* não é possível aos *testers* combinar condições de entrada, sendo que na técnica *cause-and-effect graphing* é possível combinar condições e derivar daí casos de teste muito mais eficientes. Contudo, com esta técnica podemos estar sujeitos a uma explosão de possíveis combinações. O ideal será combinar mais do que uma técnica na especificação de casos de teste [20].

Ao contrário do teste baseado em especificação, o teste baseado em implementação ou teste estrutural ou de caixa-branca, requer a inspecção do código fonte e a selecção de casos de teste que cheguem a partes do código e não são dependentes da especificação [25]. Existe uma série de limitações e desvantagens decorrentes das limitações inerentes às actividades de teste estruturais enquanto estratégia de validação [24]. Os critérios e requisitos surgem essencialmente a partir das características de uma particular implementação em teste. Nesta abordagem é avaliado o comportamento interno dos produtos de *software*, e os aspectos de implementação são fundamentais na escolha dos casos de teste. O teste estrutural baseia-se no conhecimento da estrutura interna da implementação, e tem como principal objectivo determinar se todos os elementos lógicos e de dados nos componentes de *software* estão a funcionar adequadamente. Quem testa define os casos de teste para determinar se existem defeitos na estrutura do programa. O conhecimento necessário para esta abordagem é adquirido normalmente na fase de concepção do ciclo de vida do *software*[13] [8] [24]. Estes aspectos introduzem sérios problemas na automatização do processo de validação de *software*. Independentemente dessas desvantagens, esta técnica é vista como complementar à técnica funcional e as informações obtidas pela aplicação desses critérios têm sido consideradas relevantes para as actividades de manutenção, depuração e geração de confiança no *software* [8] [24]. Esta abordagem é essencialmente usada para a realização de testes unitários.

Também categorizada por alguns autores como abordagem surge a abordagem baseada em erros, os critérios e requisitos de teste surgem do conhecimento sobre erros típicos cometidos no processo de desenvolvimento de *software* [26] [28].

É importante salientar que as técnicas de teste devem ser vistas como complementares e a questão está em como utilizá-las de forma a exponenciar as vantagens de cada uma, para que sejam melhor exploradas numa estratégia de teste, para levar a uma actividade de teste de boa qualidade, ou seja, eficaz e de baixo custo. As técnicas e critérios de teste fornecem ao programador uma abordagem sistemática e teoricamente fundamentada, além de constituírem um mecanismo que pode auxiliar a avaliação da

[13] O ciclo de vida de *software*, designa todas as etapas do desenvolvimento do software, da sua concepção ao seu desaparecimento. A primeira etapa de análise de requisitos pretende descrever de forma clara as funcionalidades do *software*, dados e informações a utilizar, resultados esperados, etc. A fase de concepção define as características do *software*, nomeadamente, estruturas de dados necessárias e detalhes dos procedimentos. A terceira fase, de implementação é o processo de tradução dos documentos das fases anteriores para código. A fase de testes é um elemento crítico para assegurar a qualidade do programa. A fase de manutenção pode introduzir alterações no *software* devido a erros encontrados ou requisitos do cliente. Com o decorrer do tempo, quase todos os programas ficam obsoletos sendo esta última etapa do seu ciclo de vida [27].

qualidade e da adequação da actividade de teste. Critérios de teste podem ser utilizados tanto para auxiliar na geração de conjunto de casos de teste como para auxiliar na avaliação da adequação desses conjuntos [25].

A estratégia de "caixa preta" pode ser usada tanto para componentes de *software* grandes como pequenos. Os testes de "caixa branca" são mais apropriados para testar componentes pequenos (pelo facto do detalhe requerido para o desenho do teste ser muito elevado) [8] [20]. A figura 3.5 demonstra as principais diferenças entre os testes de caixa preta e os testes de caixa branca.

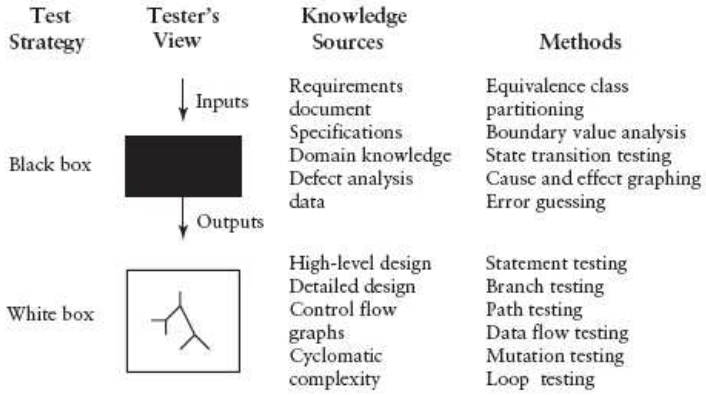

Figura 3.5 – Diferença entre testes funcionais e estruturais [8]

3.1.4. Execução de testes

Os testes podem também ser classificados quanto à sua forma de execução. Os testes manuais, são realizados directamente pelos profissionais envolvidos (*tester*), e todas as etapas e itens são avaliados por esses profissionais. Assim, são necessários recursos humanos para a execução de todos os casos de teste que foram especificados. A principal característica deste tipo de teste é o custo envolvido e a sua lenta execução. Dependendo do tamanho do sistema, dos recursos disponíveis na organização e do cronograma do projecto, a adopção dos testes manuais não é uma estratégia viável. Os testes automatizados diferem dos testes manuais, pois são executados por meio de ferramentas apropriadas para realizar um determinado objectivo. Essas ferramentas dispensam recursos humanos para outras tarefas [20].

3.1.5. Condição de execução de testes

Como já foi dito anteriormente na literatura, encontram-se várias classificações para testes de *software* uma delas é quanto à sua condição de execução em que o *software* se encontra. Assim, os testes podem ser classificados em estáticos e dinâmicos.

O teste estático é aplicado quando o *software* não está em execução, e compreende todas as técnicas baseadas em inspecções de código [24]. As técnicas mais comuns são as inspecções de código, *walkthroughs* [14] e traçagens, quando o código é analisado com o objectivo de encontrar erros. A análise

[14] Processo de inspecção de algoritmos e código-fonte, seguindo o fluxo dos algoritmos ou código, conforme determinado pelas condições de entrada e as escolhas feitas ao longo desse fluxo [30].

estática ou teste estático, segundo *Peters* e *Pedrycz* [29], divide-se em duas classes. As técnicas formais, das quais fazem parte as provas de correcção e execução simbólica. Nas provas de correcção, o código é comparado com a sua especificação, e na execução simbólica, os valores numéricos são substituídos por símbolos, procurando-se simular a execução do programa no computador. As técnicas informais, são divididas em inspecções e revisões de código analisando o projecto ou parte do código.

Os testes dinâmicos contemplam técnicas que objectivam analisar a estrutura e a funcionalidade de um *software* aquando da sua execução. São exemplos disso os testes funcionais, e os testes estruturais [24].

3.1.6. Outros tipos de testes

Existem definições de testes para todos os gostos, sendo a sua classificação também muito diversificada como é possível de comprovar com esta leitura. São descritos de forma sucinta agora alguns tipos de teste que ainda não foram apresentados. O teste de acessibilidade, verifica, se a interface do utilizador fornece o acesso apropriado às funções do sistema e a navegação adequada. Estes testes também garantem que os objectos dentro da interface do utilizador funcionem de acordo com os padrões definidos pelo utilizador. Os testes de integridade da base de dados, têm como objectivo testar de forma independente a base de dados e as regras de negócio. Este tipo de teste não deve ser realizado por via da interface de utilizador definida para o acesso aos dados. Os testes de regressão, devem ser executados na versão actual do sistema sendo o foco testar as partes do sistema que funcionavam correctamente em versões anteriores. Isto é muito útil pois todas as vezes que se insere uma característica nova no sistema deve-se testar toda a aplicação novamente, pois não se sabe o impacto dessa alteração nas funcionalidades que já haviam sido testadas com sucesso. Existem dois tipos de teste de regressão, o total e o parcial. No teste de regressão total são realizados todos os testes sem qualquer excepção. Independente das mudanças que ocorreram, deve-se testar todo o sistema. Nos testes de regressão parcial**,** são realizados testes num subconjunto de casos de teste. Esse subconjunto é definido a partir do inter-relacionamento existente entre os casos de teste as alterações efectuadas. Os testes de carga ou testes de stress têm como objectivo verificar o comportamento do *software* em situações críticas. As situações críticas podem ser entendidas como aquelas que fogem ao previsto no planeamento de funcionamento do *software*, tais como, o aumento no número de transacções, volume de dados, elevado número de utilizadores, podem levar a situações de "stress". Os testes de volume, fazem um tipo de verificação onde são submetidas grandes quantidades de dados ao sistema para determinar quais os limites que causam a falha do *software*. Este tipo de teste também identifica a carga ou volume máximo persistente que o sistema pode suportar num dado período. O teste de instalação possui dois propósitos, garantir que o *software* pode ser instalado sob condições apropriadas e verificar se o *software* funciona correctamente depois de instalado. Nos testes de configuração é validado o sistema em diferentes configurações de *software* e *hardware*, pois estas podem afectar o sistema desenvolvido [7] [31].

3.1.7. Algumas das principais ferramentas da área dos testes de software

Existem inúmeras ferramentas disponíveis no mercado com vários tipos de licença que se ocupam exclusivamente desta área de testes de software.

- **Junit** – é uma aplicação *open-source*, com suporte à criação de testes automatizados em *Java*, e serve para verificar se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas [36];
- **Selenium** – é um sistema portátil de testes de *software* para aplicações *web*. Grava os *clicks*, entradas do teclado, e outras acções para fazer um teste, que pode ser reproduzido num *browser*. Estas acções são gravadas e depois reproduzidas quando o utilizador o desejar [37].
- **Jmeter** – é uma ferramenta utilizada para testes de carga [38].
- **Clover** – é um produto comercial, mas com uma licença *open source* para projectos e instituições sem fins lucrativos. Entre as principais características deste *software* de testes incluí um histórico de relatórios, e permite testar apenas as diferenças entre uma evolução de *software* [39].
- **Watir** - *Web application testing in ruby*, é uma aplicação criada com o objectivo de permitir a automatização de testes em aplicações e páginas *web*. A abordagem é feita directamente com a manipulação do *browser*, simulando uma navegação do utilizador pela página [40].
- **Pluto** – foi desenvolvido para realizar testes unitários de forma a preencher a lacuna da *Oracle* nesta área. Funciona sobre a sua linguagem proprietária PL/SQL [41].

Este capítulo aborda sinteticamente o conceito de testes, muito mais havia a dizer, sendo aqui apresentado apenas os conceitos mais relevantes da área e de maior interesse para o trabalho realizado.

3.2. Qualidade de Software

Primeiramente, cabe destacar o que significa o termo qualidade. De acordo com o Dicionário *American Heritage Dictionary*, qualidade "é uma característica ou atributo de alguma coisa".

Para *Schmauch* [33] , uma maneira da organização alcançar qualidade num produto ou serviço é por meio da qualidade do seu desenvolvimento, e aqui os testes de *software* são essenciais. Os testes são um meio indiscutível para alcançar um *software* de qualidade.

A adopção de qualquer modelo de gestão da qualidade de *software* deve fazer uso de ferramentas que em si, avaliem se o modelo está a ser bem implementado ou não. Os modelos de gestão de qualidade apresentam algumas características importantes, tais como, documentação, rastreabilidade e padronização de processos. Entretanto, a adopção dessas características, mesmo em níveis de atendimento de expectativa muito grandes não afastam a necessidade da adopção de testes a serem aplicados às várias

fases do *software*. Este é um produto desenvolvido para a utilização em processos que pode apresentar falhas, defeitos e inconsistências como quaisquer outros produtos. Também por esta razão, é necessário recorrer a testes.

Especificamente no que se refere à indústria de *software*, sob o ponto de vista da norma ISO (*International Organization for Standardization*) 8402:2000[15], pode-se definir sistema de qualidade como um conjunto composto pela estrutura organizacional, pelos procedimentos, pelos processos e pelos recursos necessários para implementar a gestão da qualidade numa organização. Por sua vez, a gestão da qualidade pode ser descrita como o conjunto de actividades que determina a política da qualidade, os objectivos e as responsabilidades, implementando-os por meios como o planeamento, o controlo, a garantia e a melhoria da qualidade [32]. De forma simplificada, pode-se dizer que um sistema de qualidade baseado na série de normas ISO 9000[16] é um conjunto de recursos e regras estabelecidas e implementadas de forma adequada, com o objectivo de orientar cada parte da organização para que execute um conjunto de actividades de forma harmónica, de maneira correcta e no tempo apropriado, com o objectivo de tornar a organização competitiva [32]. No contexto do desenvolvimento de *software*, qualidade pode ser entendida como um conjunto de características a serem satisfeitas num determinado grau, de modo a que o produto de *software* atenda às necessidades explícitas e implícitas dos seus utilizadores [33].

3.2.1. Garantia da qualidade de *software*

A SQA (garantia da qualidade de *software* ou, em inglês*, software quality assurance*) é o conjunto de actividades necessárias para criar um acordo em que processos são estabelecidos e continuamente aprimorados tendo em vista a produção de *software* que atenda às especificações acordadas. Controlo de qualidade é o processo pelo qual a qualidade do produto é comparada com padrões aplicáveis e alguma acção é tomada quando uma não conformidade é encontrada [34]. Há também a definição do termo SQA segundo a norma IEEE 610 [22], que diz que a "garantia da qualidade de *software* é um padrão planeado e sistemático de acções para garantir uma confiança adequada que um item ou produto está em conformidade com os requisitos técnicos estabelecidos".

Segundo *Galin*, quando não se segue padrões determinados pela SQA ou há uma ausência de actividades que se relacionam com a SQA, frequentemente o *software* produzido é de baixa qualidade, contendo as seguintes características [35]:

- O *software* entregue frequentemente tem falhas;
- Consequências inaceitáveis decorrentes de falhas do sistema, desde prejuízos financeiros e perda de vidas;
- O sistema está constantemente indisponível para o propósito que foi criado;

[15] Define o sistema da qualidade como um conjunto composto pela estrutura organizacional, pelos procedimentos, pelos processos e pelos recursos necessários para implementar a gestão da qualidade numa organização [42].

[16] Designa um grupo de normas e técnicas que estabelecem um modelo de gestão da qualidade para organizações em geral, qualquer que seja o seu tipo ou dimensão [42].

- Evoluções do sistema são caras;
- Custos de detecção e remoção de defeitos no *software* são altos.

A garantia da qualidade de *software* é um esforço planeado que garante que um *software* cumpra os critérios citados acima e que possua atributos específicos do projecto como portabilidade, eficiência, reutilização e flexibilidade. Tudo isto não é responsabilidade apenas do grupo que cuida da garantia de qualidade de *software*, mas sim de um consenso entre o gestor do projecto, o líder do projecto, os programadores e os utilizadores, ou seja, dos *stakeholders*. Um trabalho típico de garantia da qualidade de *software* abrange 6 dimensões [34]:

- Métodos e ferramentas de construção;
- Revisões formais;
- Estratégia de teste;
- Controlo de documentação e histórico de alterações;
- Procedimentos para garantir a adequação aos padrões de desenvolvimento;
- Mecanismos de medição e análise.

Dependendo das características da organização, do seu processo de desenvolvimento, da maneira como ela conduz os projectos de *software*, das actividades de manutenção do *software*, e a equipa de profissionais, cada um dos componentes pode estar presente de maneira clara ou até estar ausente.

Vários modelos de qualidade de *software* fazem um estudo e definem os artefactos, os processos e as actividades que devem estar presente num sistema de garantia da qualidade. Dentro destes é de citar o CMMI (*Capability Maturity Model Integration*) [43], conhecido internacionalmente como um modelo de sucesso para quem se preocupa com SQA.

Lewis [34] categoriza os componentes mais comuns do SQA, como: Testes de *software* (verificação e validação), gestão de configuração de s*oftware* e controlo de qualidade. Na Figura 3.6, são constatadas as relações entre esses três principais componentes, e padrões, procedimentos, convenções e especificações [34].

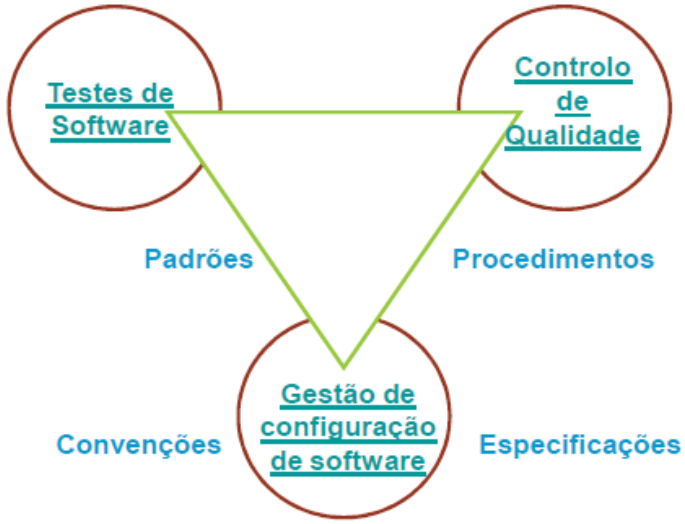

Figura 3.6 – Componentes do SQA [34]

Os testes de *software* são definidos por *Lewis* [34] como a estratégia mais popular de gestão de risco. É usado para verificar se os requisitos funcionais e não-funcionais foram devidamente implementados. Algumas limitações dos testes devem-se ao facto de muitas organizações ainda possuírem modelos de desenvolvimento, como por exemplo, o modelo em cascata, em que actividade de teste só é executada no final do processo de desenvolvimento e caso seja encontrado algum defeito (erros de requisitos, por exemplo) todo ciclo deverá ser inicializado novamente. Como é espectável nem todos os defeitos são descobertos durante os testes. Os testes de *software* focam-se também nas actividades de verificação e validação.

O controlo da qualidade é definido como um processo e um conjunto de métodos usados para monitorar o trabalho e observar se os requisitos estão a ser cumpridos. O foco é justamente em revisões e remoção de defeitos antes da entrega dos produtos. O controlo da qualidade deve ser de responsabilidade do departamento que produz o produto. O controlo da qualidade consiste em *checklists* do produto bem definidos que sejam especificados dentro do plano de garantia de qualidade. Um exemplo clássico de controlo de qualidade é a inspecção ao código. O controlo da qualidade é projectado para detectar defeitos e corrigir esses defeitos encontrados, enquanto a garantida da qualidade é orientada através da prevenção de defeitos.

A gestão de configuração de *software* é responsável por "etiquetar", rastrear e controlar as mudanças nos elementos do *software* ou do sistema. A gestão de configuração de *software* controla a evolução do *software*, gerindo as versões dos componentes do mesmo e os seus relacionamentos. O objectivo é identificar todos os componentes do *software* inter-relacionados e controlar a sua evolução através das fases do ciclo de vida de desenvolvimento de *software*. Controla o código e documentos associados de tal maneira que o código final e a sua descrição sejam consistentes e representem aqueles itens que eventualmente foram revistos e testados [34].

Para que estes três principais componentes funcionem correctamente, o sucesso do programa de garantida da qualidade de *software* também depende de uma coerente escolha de padrões, procedimentos,

convenções e especificações, conforme figura 3.6. À combinação dos componentes e as suas melhores práticas denomina-se de *Software Quality Assurance*.

3.2.2. *Capability Maturity Model*

Uma forma mais específica de procura da qualidade em processos de *software* é dada pelos modelos CMM (*Capability Maturity Model*) formulados pelo SEI (*Software Engineering Institute)* da *Carnegie Mellon University*. O primeiro CMM desenvolvido pelo SEI foi o SW-CMM (CMM *for Software*), focado apenas na disciplina da engenharia de *software*. Depois desse, outros CMM's foram desenvolvidos, focados noutras disciplinas, como engenharia de sistemas, subcontratação, pessoas e desenvolvimento integrado de produtos.

Estes modelos propõem níveis de maturidade para o enquadramento das empresas relativamente a aspectos envolvidos em projectos de *software*, como o desenvolvimento, o recrutamento e a gestão dos profissionais envolvidos. Os cinco níveis de maturidade de uma empresa que desenvolve *software*, segundo a abordagem CMM original, são [43]:

- **Inicial** (nível 1) – o processo de *software* é considerado *ad hoc,* há poucos processos definidos, e o sucesso depende basicamente de esforços, dedicação e competências individuais e "heróicas" do pessoal. É difícil prever prazos, recursos e a qualidade final do produto de *software*. Os maiores problemas encontram-se a nível da organização e gestão.

- **Repetíveis** (nível 2) – Os projectos da organização têm assegurado que os requisitos e os processos são controlados, executados e medidos. Os processos básicos de gestão de projectos são estabelecidos para controlo dos cronogramas e de custos. A disciplina garante a repetição dos processos implementados com sucesso em projectos de características similares.

- **Definido** (nível 3) – os processos de *software* para gestão e engenharia são documentados, padronizados e integrados para toda organização. Todos os projectos utilizam uma versão aprovada dos processos padrões da organização para desenvolver e manter o seu *software*. Os processos de gestão de cada aspecto do trabalho são compreendidos e executados por todos, sendo que os vários sub-processos encontram-se organizados de modo consistente garantindo o fluxo de trabalho.

- **Gerido** (nível 4) – medidas dos processos de *software* e qualidade dos produtos são avaliadas no decorrer do projecto de *software*. A organização e os projectos estabelecem objectivos quantitativos para a qualidade e performance processual e usam-nos como critério no controlo dos processos. Este nível permite o diagnóstico e análise rigorosa de variações e o seu impacto.

- **Optimizado** (nível 5) – contínuo aperfeiçoamento dos processos por toda a organização, baseados num entendimento quantitativo das causas comuns das variações inerentes aos processos. Isto é permitido pelo *feedback* quantitativo do processo e por ideias e

tecnologias inovadoras testadas na organização com o objectivo de serem detectados e corrigidos problemas.

3.2.3. *Capability Maturity Model Integration*

Para solucionar os problemas gerados pelo surgimento de outros CMM's e de normas não contempladas, o *Software Engineering Institute* criou uma nova versão do CMM, chamada CMMI (*Capability Maturity Model Integration*), com vista a resolver estes problemas [22].

O objectivo do CMMI é integrar os diversos CMM's numa estrutura única, todos eles com a mesma terminologia, processos de avaliação e estrutura. Além disso e, naturalmente, incorporar no CMMI as melhorias sugeridas pelo SEI [22] que apareceram ao longo dos anos.

O CMMI é um guia desenvolvido pela comunidade de *software* sendo um modelo único que integra os CMM's também. Incorpora as necessidades de melhorias identificadas pelo uso do CMM no mundo, é compatível com a norma ISO/IEC (*International Electrotechnical Commission*) 15434 [24] e é alinhado com o PMBOK (*Project Management Body of Knowledge)* [43]. Além disso, possui uma directriz clara e objectiva para a interpretação das práticas, apresentando sub-práticas e produtos típicos de trabalho para cada prática. Está dividido em duas representações, a contínua e por etapas, que possuem níveis de capacidade e maturidade, respectivamente. Na figura 3.7 a seguir é ilustrado os níveis para cada representação.

Nível	**Contínua (Capacidade)**	**Por etapas (Maturidade)**
Nível 0	Incompleto	
Nível 1	Realizado	Inicial
Nível 2	Repetível	Repetível
Nível 3	Definido	Definido
Nível 4	Gerido	Gerido
Nível 5	Optimizado	Optimizado

Figura 3.7 - Níveis do CMMI

A representação por etapas pode ser utilizada para verificar o nível de maturidade da organização em geral [29] . Cada área de processo pertence exclusivamente a um dos cinco níveis de maturidade. Para obter a certificação num determinado nível, a organização precisa de respeitar os passos de todas as áreas de processo definidas para o nível e os passos das áreas de processo definidas para os níveis anteriores.

O modelo CMMI v1.2, CMMI-DEV (CMMI *for Development*) contém 22 áreas de processo. Na sua representação por etapas, as áreas são divididas da seguinte forma:

Nível 1: Inicial - Não possui áreas de processo.

Nível 2: Repetível - *Requirements Management, Project Planning, Project Monitoring and Control, Supplier Agreement Management, Measurement and Analysis, Process and Product Quality Assurance, Configuration Management.*

Nível 3: Definido - *Requirements Development, Technical Solution, Product Integration, Verification, Validation), Organizational Process Focus, Organizational Process Definition,*

Organizational Training, Integrated Project Management, Risk Management, Decision Analysis and Resolution.

Nível 4: Gerido quantitativamente - *Organizational Process Performance,Quantitative Project Management*.

Nível 5: Optimizado - *Organizational Innovation and Deployment*, *Causal Analysis and Resolution*.

A estrutura desta representação do modelo é composta por cinco níveis de maturidade. Cada nível de maturidade é composto de áreas de processo, compostas por objectivos específicos e genéricos.

A representação contínua pode ser utilizada para verificar o nível de capacidade dos processos, cada área de processo possui características relativas a mais de um nível. Assim, uma área de processo que, na representação por etapas, pertence exclusivamente ao nível dois, no modelo contínuo pode ter características que a coloquem em outros níveis, ou seja, cada área de processo é classificada separadamente. À medida que as metas específicas e genéricas são atingidas para uma área de processo num determinado nível de capacidade, os benefícios da melhoria de processos são obtidos. Os níveis de capacidade focam-se no aumento da capacidade da organização em executar, controlar e melhorar o seu desempenho numa área de processo. Além disso, os níveis de capacidade permitem monitorar, avaliar e demonstrar a evolução de uma organização na melhoria dos processos associados a uma área [43].

3.2.4. Relação entre qualidade, teste e métricas

Como sugestão para uma boa política de qualidade a ser adoptada numa organização de desenvolvimento de *software*, é necessária a adopção de instrumentos que garantam um modelo de qualidade para a própria empresa. A utilização de testes para diminuir/eliminar falhas ou inconsistências já na fase de desenvolvimento, e a utilização de métricas para avaliar todo o processo, como sustentação da política de qualidade de *software*, conforme é apresentado na figura 3.8

Figura 3.8 – Importância de processos, testes e métricas para a qualidade de *software*

A utilização de métricas justifica-se pela necessidade constante de avaliar qualquer processo com o objectivo de optimizar a produção e diminuir custos. Isto porque, mesmo numa situação onde todos os testes foram considerados satisfatórios, a empresa deve-se preocupar com os custos dessa análise e, sobretudo, com o tempo gasto para atingir esse nível de qualidade. Assim, sugere-se que a qualidade de *software* seja obtida a partir do relacionamento harmónico entre os modelos, testes e métricas. O modelo**,** diz respeito à definição de etapas, fases e processos. Essa área tem uma abrangência elevada na medida em que estabelece as directrizes do projecto, bem como seus aspectos operacionais. Os testes têm o objectivo de verificar a conformidade do produto com os seus requisitos, além de primar pela procura de falhas. As métricas, têm como função medir os factores específicos do projecto para contribuir nas decisões de gestão [32].

3.3.Automatização de Testes de Software

A expressão de automatização de testes é normalmente usada quando existe um *software* que controla a execução dos testes, existem comparações entre resultados obtidos e resultados esperados, existem testes com pré-condições bem como outros controlos e funções de *reporting*.

Um *software* deve ser testado para se ter a garantir que o mesmo trabalhará como deve. O teste de *software* precisa de ser eficaz, tem de encontrar defeitos, mas deve também ser eficiente, executando os testes tão rapidamente e ao menor custo possível [7]. Para *Fewster* e *Graham* [44], automatizar os testes pode reduzir significativamente o esforço requerido para certos tipos de testes, ou aumentar significativamente o número de testes que podem ser feitos em tempo limitado. Uma estratégia madura de automatização de testes possibilitará que testes sejam executados, por exemplo, durante a noite, ou fora do horário de trabalho, em máquinas que naquele momento não estariam a ser usadas. Os testes automatizados são passíveis de repetição, usando exactamente os mesmos *inputs*, com o mesmo tempo de repetição, algo que não pode ser garantido com o teste manual.

Até mesmo as menos significativas alterações de manutenção que sejam necessárias fazer no código do *software*, podem ser fácil e rapidamente testadas de novo. A automatização elimina também muito trabalho manual. Quanto mais penosa for a execução do teste, mais se recomenda a utilização de uma ferramenta para automatização [44]. Uma abordagem referenciada não só por *Fewster* e *Graham* [44], mas também por *Zallar* [43], é a clara separação entre dois perfis de profissionais envolvidos na actividade de teste. Primeiramente, existe o *tester*, que é a pessoa responsável por gerar os melhores casos de teste, ou seja, identificam o menor número de casos de teste necessário para encontrar o maior número de erros possível. O desenhador de testes é o profissional que constrói e mantém os artefactos associados ao uso de uma ferramenta de execução do teste. É importante salientar que os dois perfis não são excludentes, e sim complementares, pois os casos de teste que um bom *tester* gera serão utilizados como início do trabalho do desenhador do teste, que fará os *scripts* que automatizam a execução dos testes. Isto não quer dizer que *testers* não possam ser bons desenhadores, mas *que os dois perfis exercitam competências diferentes. Fewster e Graham concluem afirmando que "a capacidade de testar não é somente assegurar-se de que os casos do teste encontrem uma proporção elevada dos defeitos, mas também de se assegurar que os exemplos do teste estejam bem definidos para evitar custos excessivos"*.

Para *Fewster* e *Graham*, a automatização de testes pode permitir que algumas tarefas de teste sejam executadas de maneira mais eficiente do que se estivessem a ser feitas manualmente. Segundo os autores, os benefícios são os seguintes [44]:

- **Execução de testes existentes (de regressão) numa nova versão do produto de *software*:** esta é talvez a tarefa com mais valor acrescentado pela automatização, particularmente num ambiente em que muitos programas são modificados frequentemente. O esforço envolvido para executar um conjunto de testes de regressão deve ser mínimo, dado que os testes existem e foram automatizados para funcionar numa versão mais avançada do programa, portanto deve ser possível seleccionar os testes e iniciar a sua execução com apenas alguns minutos de esforço manual.

- **Executar mais testes com mais frequência:** Um benefício claro da automatização é a capacidade de executar mais testes em menos tempo e consequentemente de tornar possível executá-los com mais frequência. Isto trará uma sustentabilidade maior ao *software*. A maioria das pessoas supõe que executarão os mesmos testes mas mais rapidamente com a automatização. Na verdade, elas tendem a executar mais testes, e esses testes são executados mais frequentemente.

- **Executar testes que seriam difíceis ou impossíveis de fazer manualmente:** Tentar executar um teste real de larga escala num sistema *online* com mais ou menos 200 utilizadores pode ser impossível, mas a se existir um input de dados que simule 200 utilizadores pode ser feito usando ferramentas de testes automatizados. Ao testar manualmente, os resultados previstos incluem tipicamente as coisas óbvias que são visíveis ao *tester*. Entretanto, há atributos que devem ser testados que não são fáceis de se verificar manualmente. Por exemplo, um objecto gráfico da interface com o utilizador pode provocar algum evento que não produz nenhuma saída imediata. Uma ferramenta de execução de teste pode certificar-se de que o evento seja provocado, o que não seria possível de verificar sem usar uma ferramenta.

- **Optimizar o uso dos recursos:** Automatizar tarefas manuais e muito morosas, tais como repetidamente digitar os mesmos *inputs d*e teste, podem ser executados com uma maior exactidão e sem recursos a pessoal.

- **Consistência e repetibilidade dos testes:** Os testes que são repetidos automaticamente serão repetidos todas as vezes da mesma forma (pelo menos os *inputs* serão, as saídas podem diferir devido ao sincronismo, por exemplo). Isto dá um nível de consistência aos testes que é muito difícil de conseguir manualmente.

- **Lançamento para o mercado antecipado:** Um conjunto dos testes que tenha sido automatizado uma vez, pode ser repetido muito mais rapidamente do que seria manualmente, assim o tempo de teste pode ser encurtado.

- **Confiança aumentada:** Sabendo que um conjunto extensivo de testes automatizados funcionou com sucesso, pode haver uma maior confiança que não haverá surpresas

desagradáveis quando o sistema for disponibilizado ao cliente, ou seja, existem provas (conjuntos de testes) que o produto funciona correctamente.

Há um elevado número de problemas que podem ser encontrados durante a tentativa de se automatizar testes. São apresentados de seguida os problemas mais frequentes quando se tenta aplicar a automatização de testes [44]:

- **Expectativas não realistas:** Há uma tendência optimista sobre o que pode ser conseguido com a utilização de uma ferramenta nova. É da natureza humana esperar que alguma solução proposta finalmente resolva todos os problemas que existiam anteriormente.
- **Experiencia em testes de *software* é baixa:** Se a experiência nos processos de teste for baixo, com testes mal organizados, pouca documentação ou inconsistente, e testes que não são muito bons a encontrar defeitos, automatizar um teste não é a melhor ideia. É preferível melhorar a eficácia dos testes do que melhorar a velocidade de execução dos testes.
- **Expectativa de que a automatização dos testes encontrará muitos defeitos novos:** A probabilidade que um teste encontre um defeito na primeira vez que é executado é muito maior. Se um teste já foi executado e passou, ao executar o mesmo teste outra vez, haverá muito menos probabilidade de se encontrar um defeito novo, a menos que o teste esteja a correr sobre código que foi mudado ou que poderia ser afectado por uma mudança feita numa parte diferente do *software*.
- **Sentimento falso de segurança:** Apenas porque um conjunto de testes funciona sem encontrar defeitos, não significa que não há nenhum defeito no *software*. Os testes podem estar incompletos, ou podem conter mesmo eles defeitos. Se os resultados esperados estão incorrectos, testes automatizados simplesmente nunca vão obter esses resultados e nunca serão bem sucedidos.
- **Manutenção de testes automatizados:** Quando o *software* é alterado frequentemente é necessário actualizar alguns, senão todos os testes, para que possam ser válidos e úteis novamente. O esforço de manutenção dos testes foi o que prejudicou muitas iniciativas de automatização de testes. Quando é preciso mais esforço para actualizar os testes do que voltar a executá-los manualmente, a automatização do teste deve ser abandonada.
- **Problemas técnicos:** As ferramentas comerciais da execução de testes são produtos de *software*. Como produtos de *software*, não são imunes a defeitos ou problemas de suporte. A interacção da ferramenta com outro *software*, ou as suas próprias funcionalidades, podem ser um sério problema. Muitas ferramentas parecem ideais no papel, mas simplesmente não funcionam em alguns ambientes. As ferramentas comerciais de teste são produtos grandes e complexos, e um conhecimento técnico detalhado é necessário para que se possa obter o melhor da ferramenta.

- **Questões organizacionais:** Automatizar testes não é um exercício trivial, e necessita de apoio da gestão e consequentemente gerar uma mudança na cultura da organização. É necessário alocar recursos para a escolha de ferramentas, para a formação, para experimentar e aprender o que é melhor, e para promover o uso da ferramenta dentro da organização.

Fewster e *Graham* ainda alertam sobre as limitações da automatização dos testes de *software* [44]:

- **Os testes automatizados não substituem os testes manuais:** Não é possível nem é desejável que todos os testes sejam automatizados. Haverá sempre actividades que serão mais fáceis de se realizar manualmente, ou que sejam impossíveis de serem transformadas em testes automatizados. Além disso, testes que são executados raramente ou que sejam modificados constantemente não compensam o esforço requerido para se automatizar.
- **Testes manuais inspiram mais confiança em relação à qualidade dos testes:** Os testes automatizados apenas comparam se os resultados obtidos correspondem aos resultados esperados. Por essa razão, o mais importante é garantir que esses resultados esperados tenham sido escolhidos a partir de um caso de teste de qualidade.

Através do exposto neste capítulo, podemos perceber que automatização existe para optimizar a execução dos testes, porém, não é uma actividade mágica que resolverá todos os problemas da actividade de teste. Existem vantagens e limitações, portanto deve ser feito um estudo da viabilidade e do custo-benefício para a utilização em cada projecto em questão.

Existem algumas técnicas de automatização de testes que são descritas na literatura desta área, sendo as principais as técnicas como *record & playback*, programação de *scripts*, *data-driven* e *keyword-driven*. A técnica *record & playback* consiste em, utilizar uma ferramenta de automatização de teste, gravar as acções executadas por um utilizador sobre a interface gráfica de uma aplicação e converter estas acções em *scripts* de teste que podem ser executados quantas vezes for desejado. Cada vez que o *script* é executado, as acções gravadas são repetidas, exactamente como na execução original. Para cada caso de teste é gravado um *script* de teste completo que inclui os dados de teste (dados de entrada e resultados esperados), o procedimento de teste (passo a passo que representa a lógica de execução) e as acções de teste sobre a aplicação. A vantagem da técnica *record & playback* é a sua simplicidade e ser prática, sendo uma boa abordagem para testes executados poucas vezes. Entretanto, são várias as desvantagens desta técnica pois se existir um grande conjunto de casos de teste automatizados, têm um alto custo e dificuldade de manutenção, baixa taxa de reutilização, curto tempo de vida e alta sensibilidade a mudanças no *software* a ser testado e no ambiente de teste. Como exemplo de um problema desta técnica, uma alteração na interface gráfica da aplicação poderia exigir a regravação de todos os *scripts* de teste [44] [45].

A técnica de programação de *scripts* é uma extensão da técnica *record & playback*. Através da programação, os *scripts* de teste gravados são alterados para que desempenhem um comportamento diferente do *script* original durante a sua execução. Para que esta técnica seja utilizada, é necessário que a

ferramenta de gravação de *scripts* de teste possibilite a edição dos mesmos. Desta forma, os *scripts* de teste alterados podem contemplar uma maior quantidade de verificações de resultados esperados, as quais não seriam realizadas normalmente pelo *tester* e, por isso, não seriam gravadas. Além disso, a automatização de um caso de teste similar a um já gravado anteriormente pode ser feita através da cópia de um *script* de teste e a alteração em pontos isolados, sem a necessidade de uma nova gravação. A programação de *scripts* de teste é uma técnica de automatização que permite, em comparação com a técnica *record & playback*, uma maior taxa de reutilização, maior tempo de vida, melhor manutenção e maior robustez dos *scripts* de teste. No exemplo de uma alteração na interface gráfica da aplicação, seria necessária somente a alteração de algumas partes pontuais dos *scripts* de teste já criados. Apesar destas vantagens, a sua aplicação pura também produz uma grande quantidade de *scripts* de teste, visto que para cada caso de teste deve ser programado um *script* de teste, o qual também inclui os dados de teste e o procedimento de teste.

As técnicas *data-driven* e *keyword-driven*, que são versões mais avançadas da técnica de programação de *scripts*, permitem a diminuição da quantidade de *scripts* de teste, melhorando a definição e a manutenção de casos de teste automatizados [44] [46].

A técnica *data-driven* (técnica orientada a dados) consiste em extrair, dos *scripts* de teste, os dados de teste, que são específicos de cada caso de teste, e armazená-los em ficheiros separados dos *scripts* de teste. Os *scripts* de teste passam a conter apenas os procedimentos de teste (lógica de execução) e as acções de teste sobre a aplicação, que normalmente são genéricos para um conjunto de casos de teste. Assim, os *scripts* de teste não mantêm os dados de teste no próprio código, obtendo-os directamente de um ficheiro separado, somente quando necessário e de acordo com o procedimento de teste implementado [44]. A principal vantagem da técnica *data-driven* é que se pode facilmente adicionar, modificar ou remover dados de teste, ou até mesmo casos de teste inteiros, com pouca manutenção dos *scripts* de teste. Esta técnica de automatização permite que o desenhador de testes e quem implementa os testes trabalhem em diferentes níveis de abstracção, dado que o desenhador de teste precisa apenas de elaborar os ficheiros com os dados de teste, sem se preocupar com questões técnicas da automatização dos mesmos [44].

A técnica *keyword-driven* (técnica orientada a palavras-chave) consiste em extrair, dos *scripts* de teste, o procedimento de teste que representa a lógica de execução. Os *scripts* de teste passam a conter apenas as acções específicas de teste sobre a aplicação, as quais são identificadas por palavras-chave. Estas acções de teste são como funções de um programa, podendo inclusive receber parâmetros, que são activadas pelas palavras-chave a partir da execução de diferentes casos de teste. O procedimento de teste é armazenado num ficheiro separado, na forma de um conjunto ordenado de palavras-chave e respectivos parâmetros [45]. Assim, através da técnica *keyword-driven,* os *scripts* de teste não mantêm os procedimentos de teste no próprio código, obtendo-os directamente dos ficheiros de procedimento de teste. A principal vantagem da técnica *keyword-driven* é que se pode facilmente adicionar, modificar ou remover passos de execução no procedimento de teste com uma necessidade mínima de manutenção dos *scripts* de teste, permitindo também que o desenhador de teste e quem implementa o teste trabalhem em diferentes níveis de abstracção [45].

3.4. Testes de software na ferramenta TestBeds

O projecto *TestBeds* surge da necessidade de criar uma plataforma capaz de automatizar testes e validações de *software*. A plataforma tem de ser capaz de automatizar e tratar os testes de forma a avaliar a performance dos *packages* desenvolvidos, mas também a evolução do seu funcionamento face a versões anteriormente testadas. As grandes preocupações passam por validar o conjunto de módulos da plataforma NGIN, bem como validar a retrocompatibilidade das versões dos produtos de *software*. Além disto a capacidade de realizar tanto, testes unitários como compostos é mais uma valência da plataforma.

Enquadrando o projecto na literatura apresentada nas secções anteriores a ferramenta *TestBeds* pode ser classificada quanto à sua fase de testes, como uma ferramenta capaz de realizar testes unitários e testes de integração. A ferramenta é capaz de realizar teste a apenas uma rotina isolada mas também a um encadeamento de rotinas, com dependências entre elas.

Avaliando o projecto quanto à sua forma de execução, a ferramenta *TestBeds* possibilita a execução de testes automatizados. A capacidade da ferramenta guardar uma bateria de casos de testes, permite-lhe executar diferentes testes, de uma forma rápida e com as repetições necessárias. A par destas funcionalidades uma preocupação da ferramenta é realizar, testes de regressão, validar se os desenvolvimentos em novas versões não estragaram o que já existia anteriormente e que tinha um funcionamento correcto.

Quanto ao seu critério de testes, a ferramenta *TestBeds,* classifica-se como uma ferramenta de testes de caixa preta.

4. A importância dos catálogos de produtos/serviços numa empresa de telecomunicações

Os catálogos de produtos e serviços começam cada vez mais a ter uma importância relevante nas organizações. Este capítulo tem como objectivo, explicar este conceito e enquadrá-lo no projecto *Configuration Manager*. Os catálogos de produtos e serviços podem ser definidos de uma forma simplista como uma aplicação com a capacidade de criar e manter produtos que podem ser vendidos para um mercado-alvo de clientes.

Este capítulo encontra-se organizado em seis secções. Na primeira secção é apresentada a organização focada em harmonizar soluções de sistemas de suporte à operação e sistemas de suporte ao negócio que serve de referência para as empresas de telecomunicações. Na segunda secção é apresentado o programa *New Generation Operations Systems and Software* (NGOSS) que se foca na automatização dos processos de negócios das empresas. Na terceira e quarta secções são exploradas duas partes do NGOSS o *enhanced Telecom Operations Map (eTOM)* e o *Telecom Applications Map (TAM)*, sendo que na quarta secção são apresentadas sucintamente as aplicações que compõem este mapa de aplicações. A quinta secção preocupa-se com uma das áreas da TAM o catálogo de produtos e serviços. Para terminar, é relacionado este conceito com a ferramenta desenvolvida o *Configuration Manager*.

4.1. Telecommunication Management Forum

O *Telecommunication Management Forum (TM Forum)* é uma organização mundial sem fins lucrativos focada em soluções para sistemas de suporte à operação (OSS[17]) e sistemas de suporte ao negócio (BSS[18]), do qual participam tanto operadoras de rede de telecomunicações, fornecedores de serviços, fornecedores de equipamentos, integradores de sistemas e programadores de software. Um dos objectivos deste fórum é mapear todos os processos das empresas de telecomunicações, visando uma padronização das soluções apresentadas pelos fornecedores, de modo a que as empresas possam substituir sistemas sem necessitar de uma grande customização, principalmente das interfaces, conforme ocorre hoje. Outro dos objectivos é promover a eficiência nos negócios entre todos os *players* do negócio de telecomunicações. Fornecer soluções práticas, directivas, padrões e orientações com intuito de transformar a maneira que os serviços são criados, entregues e modificados, são os principais objectivos do *TM Forum* [47].

[17] Operations Support Systems são sistemas informatizados utilizados pelos prestadores de serviços de telecomunicações. O termo OSS mais frequentemente descreve "os sistemas de rede" que lidam com a própria rede de telecomunicações, que apoiam processos como a manutenção de inventário de rede, serviços de aprovisionamento, configuração de componentes de rede e gestão de falhas.

[18] O Business Support Systems, refere-se a "sistemas de negócios" que lidam com clientes, no apoio a processos como a receber pedidos, processamento de contas e cobrança de pagamentos.

4.2. *New Generation Operations Systems and Software*

O *TM Forum* possui um programa denominado de *New Generation Operations Systems and Software* (NGOSS) que se foca na automatização dos processos de negócio. É uma estrutura acordada com o sector de telecomunicações, dirigida e gerida pelo *TM Forum* para [48] [49]:

- Modelar e automatizar os processos de negócio;
- Definir uma arquitectura do sistema;
- Definir um modelo de informação e de dados;
- Definir interfaces de interacção;
- Definir uma metodologia.

O ciclo de vida do NGOSS é baseado na metodologia SANRR (*scope, analyze, normalize, rationalize e rectify*) que define o ciclo de vida iterativo para o desenvolvimento destas soluções. Envolve todas as partes da empresa e terceiros no desenvolvimento de modo a que cada uma tenha o seu ponto de vista do NGOSS. Os principais benefícios são enquadrar as necessidades de todos, permitir mudanças e alterações sem grandes problemas, definir o papel de cada parte envolvida no processo e reduzir custos [49].

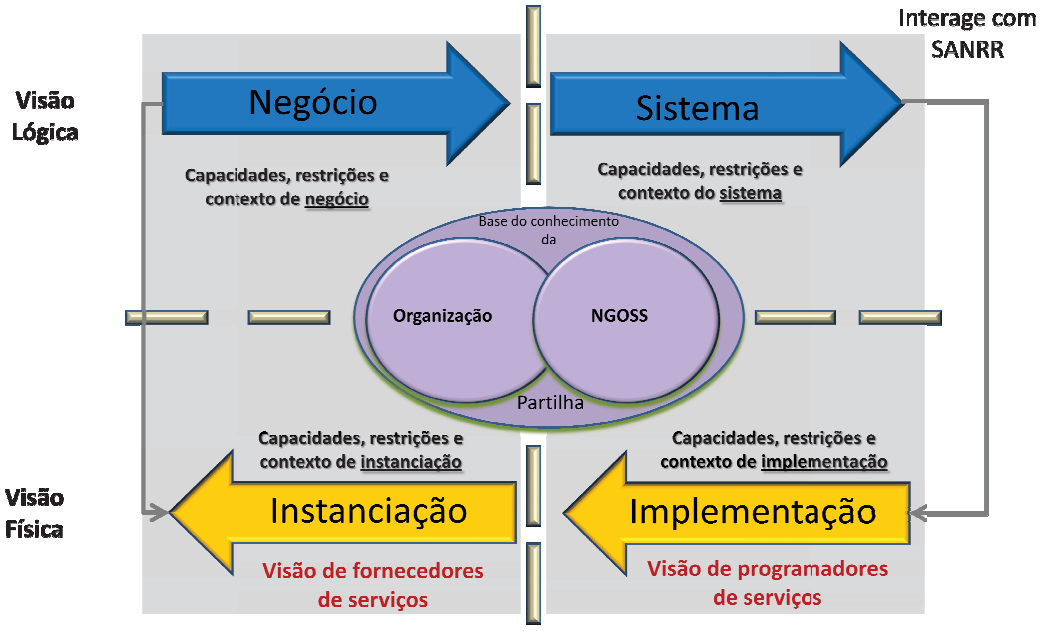

Figura 4.1 – Representação do ciclo de vida do NGOSS [49]

Como se pode verificar na figura 4.1, o ciclo de vida do NGOSS é constituído por quatro fases: na primeira são realizadas especificações no contexto de negócio, na segunda são especificados os requisitos de sistema, a terceira fase é responsável pela implementação dos sistemas e a última é responsável pela instanciação da solução. Os fornecedores de serviço participam na definição dos requisitos de negócio e da instanciação da solução para atender às suas necessidades, enquanto os responsáveis pelo desenvolvimento especificam e implementam os sistemas. O NGOSS fornece um conjunto de

metodologias para o desenvolvimento de ferramentas adequadas para as empresas de telecomunicações, de modo a obter uma visão completa do sistema, garantindo assim que todas as partes do sistema sejam integradas e forneçam o resultado esperado para a empresa.

O *TM Forum Frameworx* é constituído por quatro estruturas fundamentais, estrutura de processos de negócio (eTOM), uma estrutura de informação (SID), uma estrutura de integração de sistemas (TNA) e uma estrutura de aplicações (TAM). Estas estruturas servem como um mapa de referência para as organizações de telecomunicações, suportando o desenvolvimento e instanciação de soluções flexíveis e fáceis de integrar, e de gerir os seus ciclos de vida. A *Frameworx* define o mecanismo pelo qual o NGOSS integra os componentes numa empresa de tecnologias de informação e a sua arquitectura de processos também envolve os padrões da indústria de grandes proporções, como ITIL e TOGAF. O NGOSS é composto por quatro estruturas já apresentadas acima e agora definidas de acordo com a figura 4.2 [49]:

- A estrutura de processos de negócio, definida pelo *enhanced Telecom Operations Map* (eTOM) - Define os processos de negócios dentro e fora de uma organização de forma a padronizar uma linguagem entre as empresas. Pode ser usado para catalogar processos existentes, definir âmbitos de uma solução de software ou delimitar fronteiras claras de comunicação entre fornecedores de serviços e integradores de sistemas;
- A estrutura de informação empresarial, definida no *Shared Information and Data Model* (SID) - Oferece um modelo de informação comum e completo para todas as actividades de uma empresa. Fornece uma linguagem comum para programadores e integradores de software para descrever as informações de gestão. Utiliza UML como linguagem formal de modelação;
- A estrutura de aplicações, definida na *Telecom Aplications Map* (TAM) – é um guia para auxiliar as empresas e os seus fornecedores nas discussões sobre aplicações, de forma a tornar essas aplicações uma mais-valia para todos os interveniente. Equivalente ao eTOM para as aplicações;
- A estrutura de integração de sistemas, definida no *Technology Neutral Architecture* (TNA) - Define os princípios básicos para o desenvolvimento de uma solução baseada no NGOSS. Inclui interfaces comuns entre componentes, mecanismos de comunicação comuns, políticas e processos de gestão. Não define como programar a arquitectura, ou seja, tem uma posição neutra sobre isso.

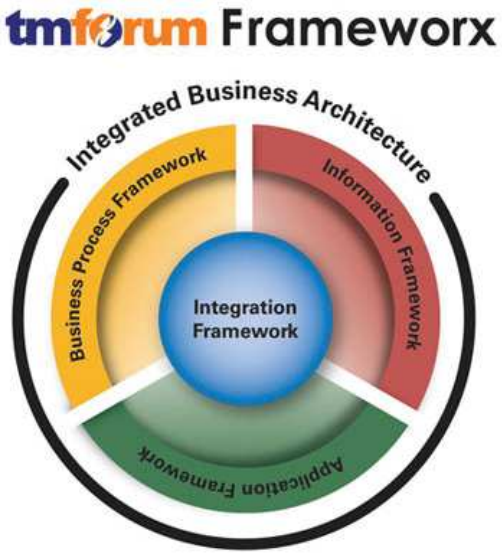

Figura 4.2 – Visão geral da Frameworx [49]

4.3. Telecom Applications Map

As TAM foram desenhadas para serem usadas por todo o espectro de *players* da cadeia de valor de *software* de telecomunicações. Podem ser usadas para uma variedade de funções e permitem que as comunidades mundiais de operadores e fornecedores, tenham um quadro de referência comum para descrever tanto as suas necessidades actuais e futuras bem como os seus requisitos. Por exemplo, um operador pode usar o mapa de modelos (tal e qual como está) das suas aplicações OSS num formato estruturado, e em simultâneo o desenvolver de um modelo futuro, conseguindo assim ter uma análise das diferenças decorrentes entre elas. Ao utilizar esta nomenclatura e *layout* comum, no panorama actual e futuro será muito mais fácil para os consultores, fornecedores ou integradores de sistemas compreenderem a situação e os seus requisitos [49].

Embora não haja uma solução categórica nesta área, as TAM oferecem um quadro de referência comum que permite aos fornecedores, consumidores e parceiros de negócio que operam nesta área, compreender os seus diferentes pontos de vista. As TAM foram construídas baseando-se em observações de sistemas típicos disponíveis hoje nos sectores do móvel, cabo e fixo e evoluirá naturalmente no desenvolvimento de novos sistemas.

Alternativamente, um fornecedor pode querer usar o mapa para destacar os sistemas que fornecem e os sistemas que tencionam fazer parcerias de forma a terem um sistema mais completo. Pode ser usado para mostrar os portfolios actuais e futuros. Por outro lado, os investidores ou analistas financeiros podem achar o mapa útil para descrever o mercado OSS em termos de crescimento, valor, etc, outros podem achar o mapa como um ponto de partida para a criação de listas de fornecedores activos em cada segmento do mapa [49].

Assim as TAM podem ser usadas em toda a cadeia de valor das telecomunicações, como é demonstrado na figura 4.3 abaixo apresentada:

Figura 4.3 *Players* da cadeia de valor das telecomunicações [49]

Sempre que possível o mapa de aplicações usa uma linguagem comum já existente na indústria e constrói um processo e um modelo de informação comum sendo essa a chave do TM Forum *Frameworx* especialmente o *Business Process Framework* (mapeado no eTOM) e a *Information Framework* (mapeado no SID). As TAM foram pensadas para serem genéricas sem perder o contacto com a realidade do mercado e serem familiares para os utilizadores da indústria, usando assim conceitos conhecidos por todos os intervenientes.

A descrição das TAM é feita seguindo uma abordagem de camadas e descreve as principais funções de cada camada, tendo cada uma delas bem definido o seu propósito. As TAM, como ilustra a figura 4.4, são segmentadas primeiramente pela *Business Process Framework* e dividem-se verticalmente nas seguintes áreas de processo: *Fulfillment*, *Assurance, & Billing* (FAB), e *Operational Support Readiness* (OSR) conjuntamente com a camada de *Information Framework* nos diferentes domínios de gestão de vendas/mercado, gestão de produtos, gestão de parcerias e fornecedores, e gestão empresarial. Existe um domínio comum a toda a estrutura de *Information Framework* que é o *Cross Domain* [49].

O mapa de aplicações também reconhece a gestão de recursos, entre eles, os recursos base de rede, plataformas de servidores de conteúdos de rede inteligente e tecnologias de controlo relacionadas com a rede, bem como infra-estruturas OSS, por exemplo tecnologia de barramentos, engenharia de gestão de processos de negócio etc.

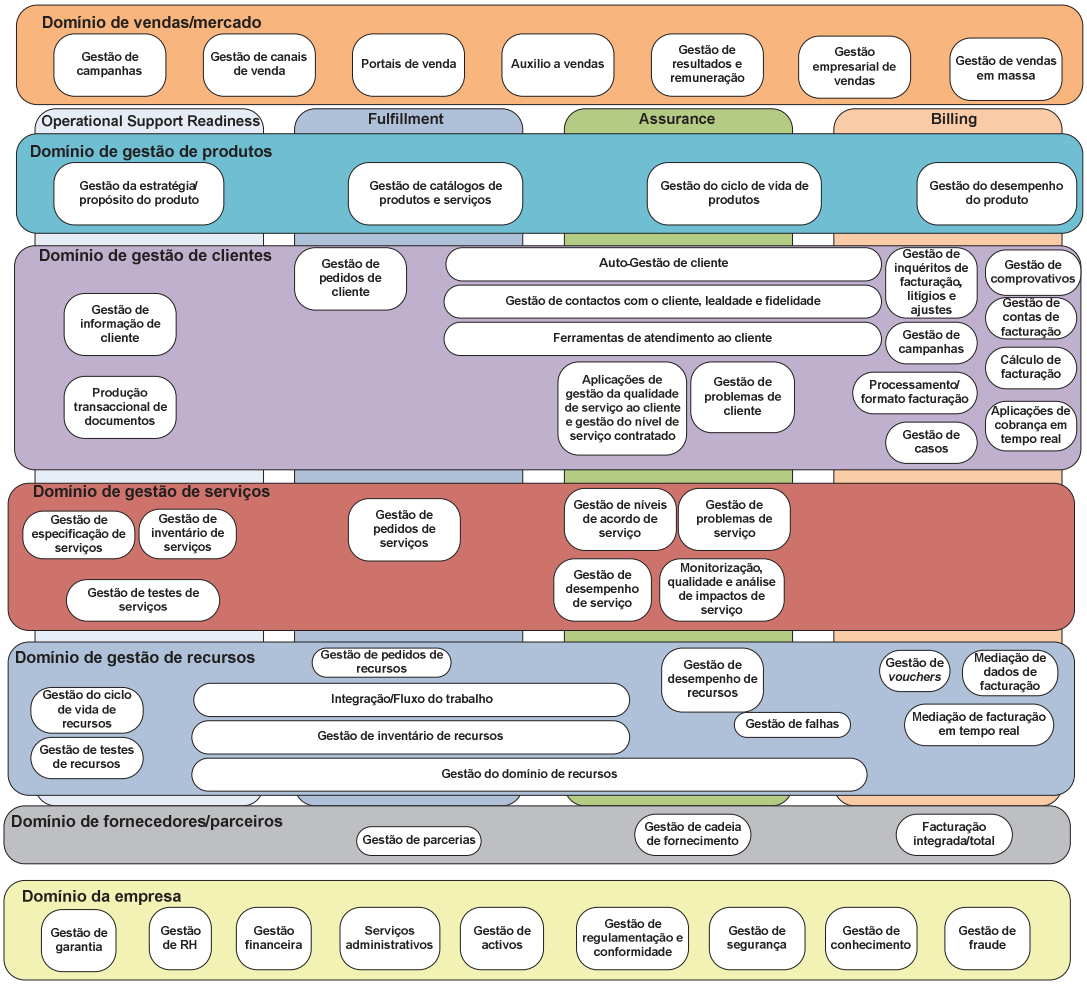

Figura 4.4 -Estrutura das TAM [49]

Apresentada a figura 4.4 ela retrata a estrutura do mapa de aplicações de referência das TAM é necessário explicar os seus domínios de forma a ser possível ter uma noção global do que as TAM podem oferecer. Esta descrição é feita em camadas e serve para ajudar os operadores e fornecedores a usarem um mapa de referência e uma linguagem comum para navegar num sistema complexo, tipicamente usado em operadoras de telecomunicações fixas, móveis ou por cabo.

Situando cronologicamente as TAM, a versão inicial concentrava-se apenas nos segmentos de operações (*Fulfillment, Assurance* e *Billing*) da *Business Framework* principalmente nas camadas de gestão de recursos, gestão de serviços e gestão de clientes. A segunda versão fornecia mais detalhes nessas camadas, bem como nas camadas de gestão de vendas/mercado, gestão de produtos, gestão de parcerias e fornecedores, e gestão empresarial. Agrupando algumas categorias da primeira versão nos segmentos *Operational Support & Readiness* (OSR), também foi introduzida a adição de contratos de suporte para algumas categorias. A versão actual evoluiu algumas aplicações essencialmente nas camadas de gestão de vendas/mercado, gestão de clientes, serviços e recursos. Além disto é introduzido o conceito de aplicações SIP (*Strategy, Infrastructure, Product*), especialmente na camada de gestão de recursos. Os conceitos de SOA foram tidos em conta também na reestruturação do domínio da aplicação [49].

Esta solução não pode ser vista como categórica, nem é algo que tenha de ser seguido integralmente, não representa um sistema infra-estrutural perfeito para um operador. O que pretende é criar junto da indústria do sector um quadro comum de referência que permite aos vários *players* como especificar, procurar, desenhar e vender operações e sistemas de suporte ao negócio e compreender os diferentes pontos de vista. De seguida são apresentados três directivas base das TAM [49].

Aplicações com uma "linguagem" comum

Uma linguagem comum para troca de informações na indústria, resultará numa redução de custos e num menor risco de investimento. O processo de aquisição será mais fácil caso se use um mapa e definições comuns de aplicações, os custos de licenciamento dos componentes serão reduzidos devido à maior reutilização e os desenvolvimentos específicos serão menores. As TAM são adoptadas pela indústria, e pelo mercado de fornecedores baseado num aumento da procura de um modelo *standard* de aplicações que beneficie os objectivos de todos.

Requisitos *standards* na aplicação

Um ponto-chave nas TAM é que a indústria tenha um padrão *standard* de requisitos das aplicações, o que vai permitir desenvolver componentes reutilizáveis, e levará a uma abordagem mais modular no desenvolvimento das aplicações. Esta reutilização vai resultar numa diminuição de custos. A par disso, a abordagem vai encorajar adopções e desenvolvimentos *standards* das interfaces entre os componentes, o que mais uma vez vai resultar numa diminuição de custos de desenvolvimento.

Disponibilizar automatização

Um padrão de componentes que resulta da adopção das TAM vai disponibilizar um maior grau de automatização entre os fornecedores de serviços deste negócio o que por sua vez irá reduzir o erro humano e incentivar a uma maior eficiência operacional. Com as soluções baseadas em mapas de aplicações *standards* é mais fácil às organizações mudar a sua forma de trabalhar adicionando e alterando componentes de suporte aos sistemas. A par disto as junções e aquisições serão mais fáceis de gerir através de uma linguagem comum das aplicações sendo também mais fácil de identificar os pontos de integração do negócio.

"Uma aplicação é um conjunto de um ou mais artefactos de software compostos por funções, dados, fluxos de negócio, regras e interfaces bem definidos", este é um dos pressupostos usados no desenvolvimento das TAM. Estes artefactos incluem [49]:

- Modelo de dados, usado entre as interfaces da aplicação;
- Politicas de gestão de aplicações internas ou externas;
- Modelo de fluxo das funcionalidades da aplicação;
- Especificações disponíveis para consultar por via de sistemas externos, as funcionalidades de interfaces da aplicação.

A definição da aplicação segundo o ponto de vista de uma arquitectura neutral é *"Uma Framework Application é um artefacto que fornece um encapsulamento de requisitos, especificações e implementações no desenho de funcionalidades, segundo as perspectivas de todos os players da cadeia de valor, necessários para apoiar nos objectivos de negócio específicos do seu ambiente operacional."*

De seguida são descritas as principais funções de cada camada/domínio das TAM, conforme apresentado na figura 4.4. O principal objectivo passa por enunciar os diferentes tipos de domínios e as suas aplicações, enquadrando assim a camada de gestão de produtos que é a mais importante para o foco deste projecto [49].

4.3.1. Aplicações com um domínio transversal

Este domínio é responsável pelas aplicações que afectam todas as outras camadas da estrutura das TAM. Os exemplos de aplicações desta camada são as de gestão de catálogos e de gestão de *fallout*. A gestão de catálogos é um domínio transversal, e a sua aplicação é feita em múltiplas camadas. Funciona como um repositório central para as entidades modularem os produtos, serviços e/ou recursos, no prazo de um ou mais domínios do ambiente de um fornecedor de serviços. A gestão de catálogos inclui a capacidade de criar e projectar novas entidades, definições de mapas de entidade, gerir regras complexas, suporte aos componentes, e gerir os seus relacionamentos e dependências [49].

4.3.2. Gestão de vendas/mercado

Esta camada é responsável por diferentes tipos de aplicações como se pode ver na figura 4.4, apresentada anteriormente, sendo de ressaltar as aplicações de gestão de campanhas, que são responsáveis por gerir o ciclo de vida de *marketing* das campanhas. Surgem da necessidade de responder às constantes mudanças dos mercados com iniciativas de *marketing* e com mensagens segmentadas para um público-alvo específico. As operadoras precisam de gerir as campanhas de uma forma adaptável e flexível de forma a poder ajustar ou evoluir os ciclos de vida de *marketing*.

Outra aplicação muito comum desta camada, é a de gestão de canais de venda, que tem como propósito fornecer a funcionalidade necessária para gerir um número específico de canais de venda. Os canais de venda estão categorizados em: vendas de campo, cujo objectivo é gerar/qualificar as oportunidades de venda, gerar receita, manter e optimizar as vendas, televendas onde a venda é feita por telefone a consumidores particulares e pequenas empresas, postos de venda fixos e móveis, concessionárias e operadores de rede virtuais. Os grandes objectivos passam por criar e promover ligações, criar e promover contactos, disponibilizar folhetos informativos, entre outros [49].

4.3.3. Domínio de gestão de produtos

O domínio de gestão de produtos está dividido em 4 pontos-chave: gestão da estratégia / propósito de produtos, gestão do catálogo de produtos e serviços, gestão do ciclo de vida do produto e gestão do desempenho do produto.

4.3.3.1. Gestão da estratégia/propósito do produto

A estratégia de produto é um plano de acção para alcançar os objectivos de uma estratégia operacional através dos produtos vendidos no mercado. O propósito do produto é visto como um conjunto de ideias sobre como a estratégia será concretizada através de produtos vendidos nos mercados-alvo específicos. A gestão da estratégia/propósito do produto é, portanto, a capacidade de capturar e gerir os detalhes da estratégia de uma empresa e o que daí resulta, escolhendo os produtos que vão desenvolver, distribuir e vender. Esta capacidade permite a gestão desta informação ao nível da empresa, através de diferentes grupos operativos e unidades de mercado na qual a empresa opera. Finalmente, fornece a capacidade de relacionar os propósitos do produto para a sua venda real a fim de controlar a forma como a estratégia de produto será até realmente ser disponibilizada no mercado. A capacidade de reter essa informação, permite um melhor desempenho de futuro na validação ou anulação de uma estratégia de produtos da empresa e futuras propostas [49].

4.3.3.2. Gestão de catálogos de produtos/serviços

A definição de produto pode ser visto como um item que satisfaça as necessidades do mercado. A gestão do catálogo de produtos/serviços é a capacidade de criar e manter produtos que podem ser vendidos para um mercado-alvo de clientes. Mais especificamente, é a capacidade de modelar explicitamente a estrutura de um produto, e em seguida, criar e gerir centralmente as suas instâncias (ou "catálogo") de produtos com base nessa estrutura. Os produtos não são sempre discretos, ou itens únicos, um produto pode ser um conjunto de componentes associados e vendidos como uma única entidade comprável. Assim, o produto pode ser composto por vários componentes, tangíveis ou intangíveis, tais como serviços, recursos, dispositivos, ofertas comerciais, etc, que são "construídos" para formar uma única entidade “vendível” [49].

Estes serviços de base e os recursos podem ser geridos por diferentes partes da organização. O domínio da gestão de produtos normalmente é responsável pela gestão dos catálogos de produtos/serviços preocupando-se com a criação e actualização de produtos utilizando os componentes disponíveis, ou seja, anteriormente criados/configurados. As principais funções das aplicações de gestão de catálogos de produtos/serviços são:

- Modelo de dados estrutural de um produto - Contém o modelo de dados completo da estrutura de entidades do produto e as relações que regem o comportamento de um produto e os seus componentes subjacentes;
- Instanciação do produto – Criar e manter instâncias do produto baseado numa estrutura comum, gerando um catálogo de produtos centralizado;
- Manutenção dos componentes do produto - Criar e manter os diferentes componentes que podem compor um produto, bem como criar e manter as relações entre os componentes para formular instâncias completas de produtos.
- Reutilização de componentes - Reutilizar componentes em diferentes instâncias do produto;

- Gestão de uma representação lógica do serviço - Criar e manter a representação lógica dos serviços;
- Gestão do relacionamento de recursos e os componentes dos serviços – Criar e manter as relações entre os componentes e os seus serviços e recursos;
- Gestão de funções externas ao produto – Fornecer uma visão do modelo completa das instâncias do produto para funções/aplicações externas, etc.
- Gestão de dados do produto em todo o *workflow* – inclui as actividades e ferramentas de gestão e manutenção de dados de todos os produtos de uma determinada organização. Os repositórios de dados são fundamentais para apoiar estas actividades, e tendem a ser repositórios de informação altamente configuráveis e são geralmente suportados por um fluxo de trabalho. A navegação e procura de capacidades são predominantes nesta área. Os grupos funcionais podem incluir (mas não estão limitados a estes):
    - Especificações do produto detalhadas;
    - Informação contratual;
    - Informação de histórico de um produto;
    - Gestão de documentos;
    - Gestão de configurações;
    - Gestão de alterações técnicas;
    - Interoperabilidade e integração de dados com o catálogo de produtos.

A figura 4.5 ilustra as categorias de dados essenciais que compõem o modelo de um catálogo de produtos/serviços e a relação dos dados com outras aplicações.

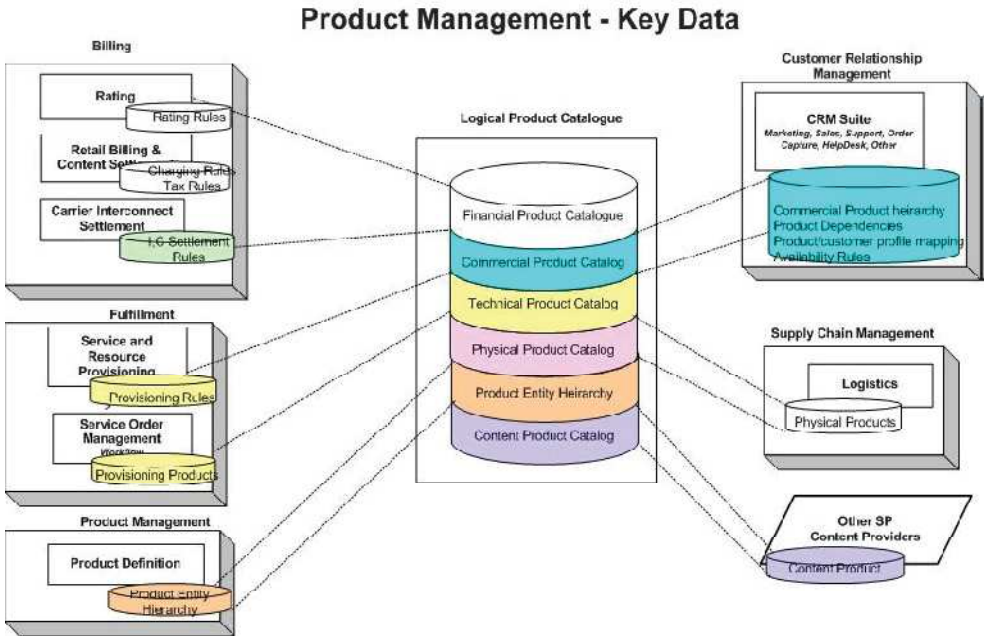

Figura 4.5 – Estrutura lógica de um catálogo de produtos [49].

As aplicações de gestão de produtos/serviços típicas podem conter as seguintes funções:

- Oferta de serviços e produtos;
- Hierarquia de negócio do produto;
- Hierarquia comercial do produto;
- As regras relativas às ofertas, incluindo os pré-requisitos e o relacionamento com outras ofertas e os seus parâmetros;
- Mapeamento do perfil do cliente/produto;
- Regras disponíveis;
- Período de validade de um serviço/produto;
- Ciclo de vida de um produto/oferta incluindo reestruturação do prazo do projecto;
- Níveis de serviço disponíveis;

Descrevendo superficialmente os diferentes tipos de catálogos, estes podem-se classificar em catálogos financeiros geralmente compostos pela avaliação de atributos, informação de tarifários, informação de liquidação, informação dos níveis de serviço, cobrança única e/ou recorrente, informação de descontos, ofertas relacionadas com questões contratuais, e custo do produto. Os catálogos técnicos geralmente contêm, formas de fornecer a informação necessária para construir o fluxo de trabalho para prestar um serviço. O catálogo físico contém normalmente o equipamento físico a ser fornecido como parte de uma oferta. O catálogo de entidades de um produto normalmente define a relação de um produto/serviço a outro produto/serviço com a finalidade de localizar os produtos através de bases de dados distribuídas. Por exemplo, inclui a hierarquias utilizando um ID de referência comum para cada um dos locais do catálogo de produtos [49].

4.3.3.3. Gestão do ciclo de vida do produto

A gestão do ciclo de vida do produto é responsável por todo o ciclo de vida de em produto e dos seus componentes. Isso inclui todos os processos necessários para projectar, construir, implementar, manter e, finalmente, retirar o produto do mercado. A gestão do ciclo de vida do produto inclui também as actividades e ferramentas usadas para definir novos produtos e actualizações de produtos existentes. Geralmente estas actividades exigem um significativo grau de colaboração, muitas vezes através de diferentes localizações geográficas. Preocupa-se ainda em ir ao encontro das necessidades/preferências do cliente e o mapeamento dos recursos de produtos em actuais e futuros [49].

4.3.3.4. Gestão de desempenho do produto

A gestão do desempenho de um produto inclui as actividades e ferramentas que extraem e analisam dados quanto à eficácia da estratégia do produto e as propostas com base no seu desempenho no mercado [49].

4.3.4. Domínio de gestão de clientes

Esta camada é responsável pelas aplicações de gestão de clientes. A gama de aplicações desta camada é bastante extensa sendo as mais usuais desta camada, as aplicações de gestão de informação de cliente que garante a entrega de informação consistente sobre um cliente, precisa e completa do ponto de vista operacional e analítico em toda a empresa e prestadores de serviços, possibilitando a optimização de processos de negócio e de modo a alcançar novas oportunidades de negócio. A informação de cliente está normalmente espalhada por ambientes mistos com fragmentação de dados do cliente, estas aplicações de gestão de informação, usam um contexto de lógica de negócios confidenciais, sincronizam as informações do cliente através de fornecedores de sistemas e harmonizam os dados de cliente evitando inconsistências. Outro tipo de aplicações são as de auto-gestão que fornecem uma interface via *Internet* para o cliente realizar uma variedade de funções de negócio directa e autonomamente. Estas aplicações interagem para oferecer um serviço inteiramente automatizado ou um serviço assistido por vários pontos de contacto com os clientes. As aplicações de auto-gestão de clientes são normalmente disponibilizadas pelo *Customer relationship management* (CRM). As aplicações de gestão de problemas do cliente têm como objectivo gerir os problemas relatados pelos clientes, resolvendo-os para que a satisfação do cliente não fique abalada, fornecendo um *status* preciso sobre o problema ao cliente. Os problemas do cliente podem incluir [49]:

- Perguntas gerais sobre os produtos adquiridos, e a forma como são usados pelo cliente;
- Problemas com os produtos já adquiridos e o seu uso, quer devido à falta de formação ou problemas de serviço/rede;
- Problemas com a compra de material do fornecedor de serviços;
- Informações gerais, reclamações e elogios.

4.3.5. Domínio de gestão de serviços

Esta camada é responsável pelas aplicações de gestão de serviços. Apresentando as aplicações, este domínio as principais são as aplicações de gestão de pedidos de serviço que permitem gerir do princípio ao fim o ciclo de vida de uma solicitação de serviço. Isso inclui validar a disponibilidade do serviço, bem como o pedido de serviço. Outra funcionalidade inclui a emissão do pedido de serviço, e/ou a decomposição de pedido do produto. As notificações são emitidas para a gestão de pedidos durante o processo de manipulação de serviços (especialmente após a sua conclusão). Outro tipo, são as aplicações de gestão de testes de serviço que estão focadas em assegurar que os vários serviços estão a funcionar correctamente, estas aplicações fazem parte do processo de garantia. No processo de atendimento, o teste de serviço é responsável por garantir que o serviço atribuído funciona como definido. Como parte do processo de teste, estas aplicações também podem ter problemas na sua interface, o que pode desencadear um teste automático [49].

4.3.6. Domínio de gestão de recursos

Esta camada é responsável pelas aplicações de gestão de recursos. É composto por um conjunto de aplicações sendo as mais comuns as de gestão de *vouchers* que controlam todos os aspectos de cupões de recarga pré-pagos. Geralmente, um *voucher* tem um único número de série e um código PIN, através da qual é identificado. Os *scratch-vouchers* podem ser comprados a partir de máquinas automáticas, quiosques e outros pontos de venda. Os clientes podem usar os *vouchers* para recarregar o seu saldo através de um sistema de *Interactive voice response* (IVR). Outro tipo de aplicações são as de gestão de falhas que disponibilizam as funções necessárias para gerir falhas associadas a recursos específicos. Isso inclui a detecção, isolamento, resolução e geração de relatórios de diversas falhas [49].

4.3.7. Domínio de parcerias/fornecedores

O domínio de gestão de parcerias/fornecedores está dividido em três pontos-chave, gestão da cadeia de fornecimento, gestão de parcerias e aplicações de facturação interligada/total. A principal aplicação deste domínio é a de gestão da cadeia de fornecimento que tem como objectivo permitir a criação de redes logísticas adaptáveis às organizações. A definição de estratégias competitivas e funcionais assegura, através do posicionamento (tanto com fornecedores, como com clientes), dentro das cadeias de fornecimento em que se inserem, o alinhamento entre a procura e a oferta. O sistema inclui processos de logística que abrangem desde a entrada de pedidos de clientes até a entrega do produto no seu destino final, envolvendo o relacionamento entre documentos, matérias-primas, equipamentos, informações, pessoas, meios de transporte, organizações, tempo, entre outros [49].

4.3.8. Domínio da gestão empresarial

Esta camada é responsável pelas aplicações de gestão de garantia, gestão de recursos humanos, gestão financeira, gestão de activos, entre outras. As aplicações de gestão de recursos humanos servem para melhorar a eficiência da força de trabalho e a sua produtividade. Planear futuras necessidades da força de trabalho, encontrar e desenvolver competências no pessoal, formar a força de trabalho para garantir que cada funcionário tem as habilidades certas e está alinhado com as estratégias empresariais e com as metas da equipa e individuais, são os seus principais objectivos. Outra aplicação importante desta camada são as aplicações de gestão de segurança que oferecem uma abordagem unificada de segurança, e as tecnologias de base para [49]:

- Sistemas de protecção e infra-estruturas de rede usados contra os acessos não autorizados;
- Restringir os serviços apenas a utilizadores autorizados;
- Evitar qualquer negação de serviço a utilizadores autorizados;
- Proporcionar as medidas necessárias para detectar, documentar e contrariar ameaças.

4.4. Catálogo de produtos e serviços na ferramenta Configuration Manager

A PT Inovação como uma empresa de referência na área das telecomunicações em Portugal está atenta a todos as formas de evoluir e melhorar a solução que neste momento tem. O *TM Forum* é tido muito em conta nos processos de análise de possíveis melhorias, e como o sua *Frameworkx* tem sempre em conta o modelo actual e o modelo futuro de uma solução, é possível na empresa pensar em evoluções e em simultâneo desenvolver as necessidades actuais. Outra vantagem que a *TM Forum Frameworkx* possuí é de não obrigar a uma alteração total da solução. Permite integrar módulos/aplicações no seu mapa de aplicações de referência de uma forma progressiva, e neste momento já existe um conjunto de áreas que estão implementadas segundo este mapa de aplicações de referência na empresa. São exemplos disso a aplicação de gestão de *vouchers* já utilizada hoje em dia, a de facturação em tempo real, gestão de campanhas, entre outras. Enquadrando o projecto na literatura apresentada nas secções anteriores a ferramenta *Configuration Manager* também está situada numa área das TAM, o CM deve ser classificado como um catálogo de produtos/serviços, com uma vertente mais focada para os catálogos técnicos, pois não é virado para a venda directa ao público. Se eliminarmos essa parte de catálogo mais comercial, podemos ver o CM como a ferramenta de gestão de catálogos técnicos de um conjunto de sistemas. Permite portanto fazer a gestão de um conjunto de entidades de configuração de sistemas, sendo o produto/serviço deste catálogo, as ofertas comerciais que são configuradas na aplicação pelas operadoras para usufruto dos seus clientes.

Na prática o *Configuration Manager* permitirá a configuração de um conjunto de regras e entidades de negócio que na sua globalidade dará origem a um produto que o cliente vai usufruir. Se usarmos um exemplo prático, uma oferta comercial pré-paga/pós-paga deve ser visto como um serviço que o cliente adquire aquando da compra de um equipamento. Essa oferta comercial por si só não é nada, o que difere esse produto de outro produto é a sua constituição, os componentes/entidades/regras que são diferentes formando assim um conjunto de novas ofertas. Como é de conhecimento de todos existe um número quase infinito de ofertas hoje em dia, ofertas comercias, com um número limitado/ilimitado de mensagens gratuitas, chamadas mais baratas para clientes da mesma operadora, chamadas com um preço fixo para todas as redes, entre muitas outras, sendo que cada uma delas é uma oferta comercial distinta, um produto/serviço do catálogo de produtos/serviços da operadora.

O CM permite exactamente configurar estes produtos, mas não só, existe um conjunto de entidades que isoladas não representam um produto mas que também têm de ser configuradas de forma a serem reutilizadas em diferentes produtos, por exemplo, um alarme de saldo baixo. Esse alarme é configurado isoladamente e todas as ofertas comerciais pré-pagas caso atinjam um valor limite, também ele configurado, são notificadas por uma mensagem informativa de que saldo é menor que x€.

Como se pode conferir é essencial para a empresa uma ferramenta que centralize e permita criar este conjunto de ofertas comerciais, daí o CM ser uma enorme mais-valia para a solução NGIN.

5. Descrição do trabalho no âmbito da ferramenta *TestBeds*

O objectivo deste quinto capítulo, é apresentar ao leitor a primeira ferramenta desenvolvida, a *TestBeds*. É apresentado o modelo de dados bem como o seu funcionamento interno, sendo também apresentadas as características desta ferramenta e a sua interface gráfica com o utilizador.

A ferramenta *TestBeds* surge da necessidade de criar uma plataforma capaz de automatizar testes e validações de *software*. Devido ao elevado ritmo de construção, evolução e manutenção de *software*, surge permanentemente a necessidade de centralizar o controlo de testes e de avaliação de desempenho para que as entregas de *software* sejam efectuadas com o mínimo de falhas e com uma elevada robustez e fiabilidade. A ferramenta *TestBeds*, pode ser usada de duas formas. Num primeiro cenário com o objectivo de executar *TestSequences*[19] sobre um determinado plano de testes, ou então ser usada para realizar testes unitários sobre unidades de testes mais pequenas (rotinas isoladas). A ferramenta *TestBeds* tem esta capacidade de fazer os dois tipos de testes que se complementam, como se pode verificar na figura 5.1.

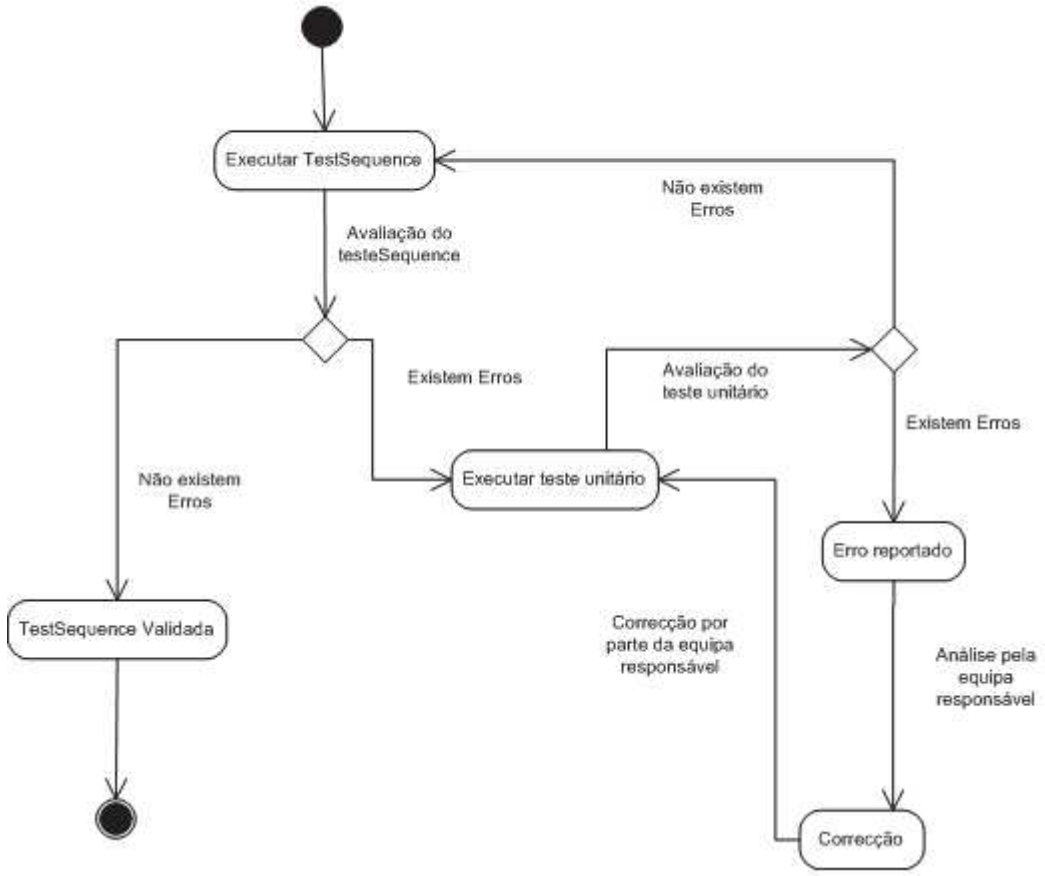

Figura 5.1 – Diagrama com o fluxo da utilização normal da ferramenta *TestBeds*

[19] Um teste composto (*TestSequences)* é um tipo de teste que permite num único teste, percorrer vários métodos, procedimentos, funções e fazê-lo várias vezes, encadeando assim o seu *workflow*. Este teste é muito útil pois permite o manuseamento de diversos procedimentos que muitas vezes estão interligados entre si, por exemplo os parâmetros de entrada de alguns procedimentos podem ser parâmetros de saída de outros..

Na prática isto significa, num cenário em que uma invocação que pertence a um teste composto não tem o comportamento que se espera, não é conclusivo que todas as invocações desse teste estejam erradas. Esta invocação terá de ser analisada posteriormente e de forma isolada, pois está com um comportamento ambíguo, logo com um teste unitário será possível descobrir mais facilmente o porquê desse comportamento. Como as *TestBeds* têm associados a si uma bateria de testes armazenada, após a correcção desse erro a *TestSequence* deverá ser novamente invocada de forma a validar se o teste composto já tem o comportamento esperado.

Os testes unitários serão definidos na prática como um teste pertencente a um plano de testes que tem apenas uma invocação, as *TestSequences* compõem um teste com várias invocações. Os motivos para a criação da plataforma passam pela:

- Necessidade de centralizar o controlo de testes;
- Ter uma bateria de testes armazenada e reutilizada facilmente;
- Existir o mínimo de falhas na entrega do *software*:
- Garantir a retrocompabilidade entre versões.

5.1.Modelo de dados

Para o funcionamento das *TestBeds* é necessário recorrer ao armazenamento de informação. Esta informação é necessária em alguns casos para o funcionamento da ferramenta, sendo que também existe informação que resulta do uso da ferramenta. Existem três tipos de tabelas, as tabelas de configuração, as tabelas de resultados (relatórios) e as *lookup tables*.

As tabelas de configuração e as *lookup tables* contêm informação mais estática. As *lookup tables*, a verde na figura, contém informação sobre o produto a ser testado, as tabelas de configuração, a azul, contêm informação criada e inserida a partir das configurações feitas na aplicação. As tabelas amarelas são tabelas de resultados, mais dinâmicas e são actualizadas sempre que se corre um teste.

O diagrama de entidade-relacionamento apresentado na figura 5.2 enuncia as entidades existentes bem como as ligações entre elas, para a versão 1.0 das *Testbeds*.

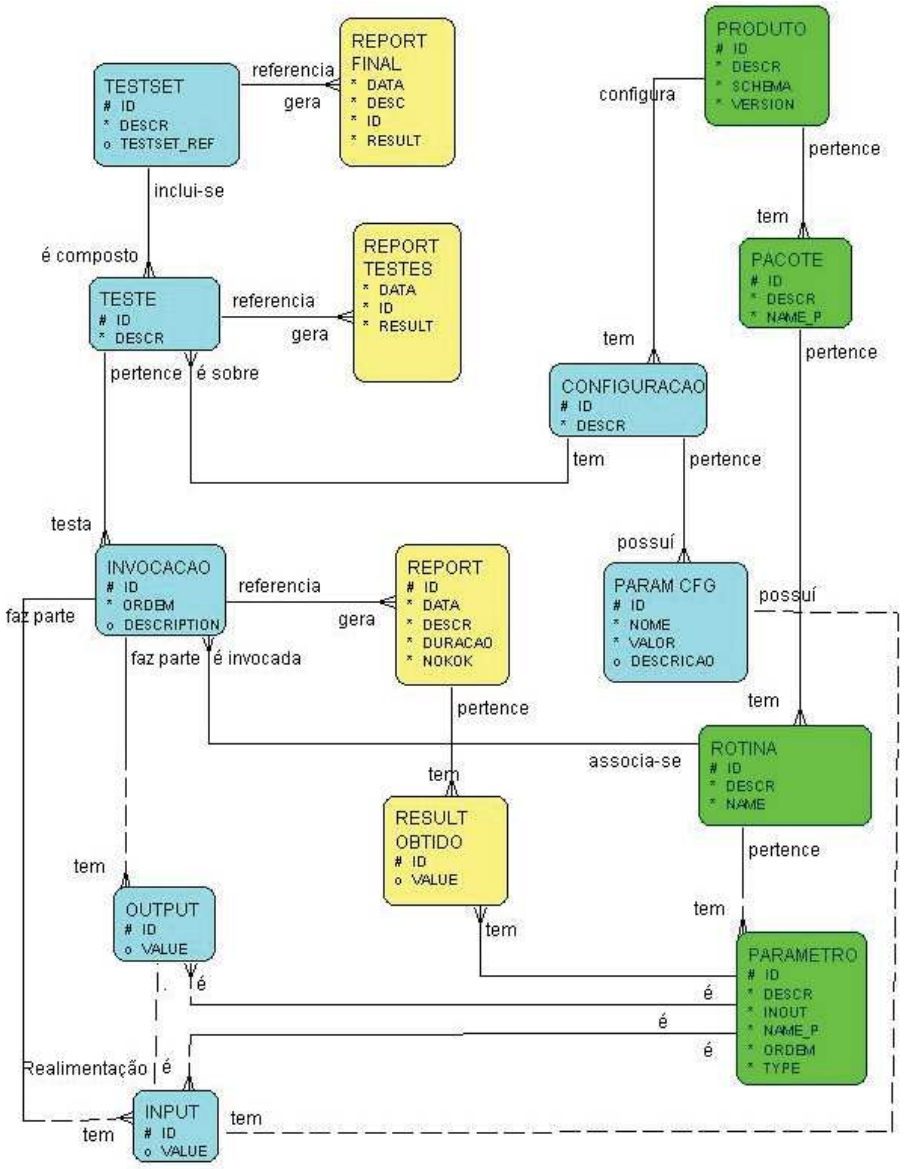

Figura 5.2 – Diagrama de Entidade Relacionamento para a versão 1.0

É apresentado de seguida a descrição da figura 5.2. Um PRODUTO é o alvo do teste, normalmente uma versão de *software*. No modelo de dados da ferramenta um produto tem um conjunto de configurações e é composto por um conjunto de pacotes. Toda a informação sobre os produtos carregados para a aplicação é aqui armazenada.

A entidade PACOTE diz respeito a todos os pacotes/subprodutos de um PRODUTO. No seu uso normal um PACOTE corresponde aos módulos da suite genérica (Alarmes, Notificações, etc). No modelo de dados um pacote pertence a um produto e é composto por um conjunto de rotinas. Os pacotes carregados de cada produto para a aplicação são armazenados nesta entidade.

Uma ROTINA pertence a um PACOTE, e é sobre esta que as invocações dos testes vão incidir. Correspondem às rotinas e procedimentos internos de cada modulo da suite genérica. Uma rotina pertence a um pacote e é constituída por um conjunto de parâmetros, tendo associado a si um conjunto de invocações. Toda a informação sobre as rotinas que compõem cada pacote é armazenada nesta entidade.

Um PARAMETRO pertence sempre a uma rotina a ser testada e pode dar lugar a um ou vários resultados obtidos. Os parâmetros podem ser *inputs* ou *outputs*, ou ambos.

Um TESTSET é um conjunto de testes de um determinado módulo/sub-módulo. Um TESTSET dá origem a um relatório do *testSet* sempre que é executado. Nesta entidade é armazenada a informação sobre os *testsets* configurados na aplicação.

Um TESTE está incluído num TESTSET e deve ser visto como o teste a uma funcionalidade/requisito. Um teste tem sempre uma configuração associada, e é constituído por um conjunto de invocações. No caso do utilizador desejar executar testes unitários, o teste deve ser composto por apenas uma invocação. Sempre que um teste é executado é gerado um relatório do teste. Nesta entidade é armazenada a informação sobre os testes configurados na aplicação.

Uma INVOCACAO está incluída num TESTE e deve ser visto como um *step* de um teste. Uma invocação pode ser constituída por *outputs* e por *inputs*. Sempre que uma invocação é testada é gerado um relatório da invocação. Nesta entidade é armazenada a informação sobre as invocações configuradas na aplicação e a sua ordem de execução dentro de um teste.

Um parâmetro de INPUT pertence sempre a uma invocação e são os parâmetros de INPUT que têm os valores a serem usados para alimentar as invocações. Um parâmetro de *input* é um PARAMETRO. Uma característica especial é que um parâmetro de *input* pode ser em simultâneo um parâmetro de *output*, ou um parâmetro de configuração.

Um parâmetro de OUTPUT pertence sempre a uma invocação e os parâmetros de OUTPUT são os resultados esperados de obter após uma invocação. Um parâmetro de *output* é um PARAMETRO.

Uma CONFIGURACAO pode ser usada por um ou mais TESTE, e pode ter associado a si vários PARAM. CFG. Uma CONFIGURACAO pertence sempre a um PRODUTO.

Um PARAM. CFG. pertence sempre a uma CONFIGURACAO. Um PARAM. CFG. pode ser um parâmetro de INPUT.

Na entidade *REPORT* FINAL são armazenados todos os relatórios dos *testSets* executados. Um *REPORT* FINAL diz sempre respeito a um *testset*.

A entidade *REPORT* TESTES armazena todos os relatórios dos testes elementares que constituem um *testset*. Um *REPORT* TESTES é sempre referente a um teste.

A entidade *REPORT* armazena todos os relatórios das invocações que fazem parte dos testes elementares executados. Um *REPORT* é sempre referente a uma invocação.

A entidade RESULT. OBTIDO armazena todos os resultados dos parâmetros de saída das invocações testadas Um RESULT. OBTIDO está sempre associado a um parâmetro de saída.

5.2. Build Serialize – O motor da ferramenta

O *Build_Serialize* é um pacote de *software* desenvolvido em PL\SQL e surge da necessidade de criar um mecanismo interno capaz de interpretar de uma forma simples os tipos não nativos de base de

dados usadas nos módulos que constituem a suite genérica. O tratamento dos atributos nativos *(Number, Varchar2)* são de fácil interpretação no entanto nas estruturas mais complexas já não é assim. O *Build_Serialize* tem então como função a criação de uma forma transparente e fácil de interpretar os valores destas estruturas pela ferramenta *TestBeds*.

Basicamente o *Build_Serialize* é um construtor de funções de *serialize*[20] e do seu inverso *deserialize*[21]. O *Build_Serialize* tem um funcionamento bastante simples, recebe o nome do tipo de dados não nativo, acede ao dicionário de dados para analisar toda a estrutura que compõe esse tipo e armazena essa informação numa expressão. De seguida essa expressão é interpretada e inserida numa estrutura de dados criada para suportar esta informação. Após o findar do preenchimento desta estrutura, esta é percorrida e analisada e a partir dessa análise são geradas as funções de serialização e "deserialização" para o tipo não nativo indicado. Estas funções vão serializar os tipos não nativos numa estrutura em XML, que depois será quando necessário "deserializada" novamente para a estrutura do tipo nativo. Este processo é transparente para o utilizador, não se apercebendo de tal comportamento. O grande objectivo é que para o utilizador seja tão transparente definir um objecto do tipo inteiro como um tipo de dados complexo. O funcionamento deste pacote de *software* pode ser descrito resumidamente pelos seguintes pontos:

- Pesquisa e identificação no dicionário de dados, da estrutura de dados com o tipo a serializar, resultando numa expressão com a assinatura e a sua composição;
- Tratamento da expressão resultante do ponto anterior e inserção numa estrutura de dados interna;
- Geração das funções de serialização/deserialização dos tipos complexos e possíveis subtipos que o compõem.

Num cenário em que todos os módulos da suite genérica estejam carregados no modelo de dados da ferramenta *TestBeds*, vão existir duas funções para cada tipo não nativo que esteja definido no ambiente em que a ferramenta está a ser executa.

[20] Quando falamos em serialização de objectos, o objectivo passa por procurar uma solução para interpretar objectos entre sistemas, aplicações e até mesmo entre redes. No caso do *build_serialize* este processo é feito interpretando o objecto que pretendemos serializar (tipo não nativo), criando uma estrutura em XML que armazene e identifique tudo o que diz respeito ao objecto. De uma forma excessiva podemos dizer que a serialização duplica o objecto, só que numa estrutura em XML.

[21] O inverso do processo de serialização, ou seja, parte-se de uma estrutura em XML, esta é analisada e interpretada, e é criado um objecto no seu formato original.

```
<NGIN_APP_COMPILED_RULE_VARRAY>
  <item>
   <index_value>1</index_value>
    <NGIN_APP_COMPILED_RULE_REC>
     <RULE_TYPE></RULE_TYPE>
     <SI_OUT1></SI_OUT1>
     <SI_OUT2></SI_OUT2>
     <NGIN_APP_RULE_VARRAY>
      <item>
      <index_value>1</index_value>
      <NGIN_APP_RULE_REC>
<RULE_VALUE>CALL_DATE</RULE_VALUE>
        <VALUE_TYPE>T</VALUE_TYPE>
      </NGIN_APP_RULE_REC>
      </item>
      <item>
       <index_value>2</index_value>
       <NGIN_APP_RULE_REC>
         <RULE_VALUE>10</RULE_VALUE>
         <VALUE_TYPE>K</VALUE_TYPE>
       </NGIN_APP_RULE_REC>
       </item>
     </NGIN_APP_RULE_VARRAY>
    </NGIN_APP_COMPILED_RULE_REC>
  </item>
</NGIN_APP_COMPILED_RULE_VARRAY>
```

Figura 5.3 – Exemplo de uma estrutura XML gerada pelo BUILD_SERIALIZE

A figura 5.3 apresenta uma estrutura XML, que foi gerada pelo *Build_Serialize*. As *tags* de XML da figura acima dizem respeito aos subtipos existentes na estrutura do tipo principal, com a excepção das *tags* <item> e <índex_value>, que são *tags* usadas para a gestão interna do pacote de *software*. Sucintamente estas duas *tags* são colocadas quando se está perante tipos como *Varrays, Nested Tables* ou *index – by tables*, e simbolizam a sua posição nessa estrutura. No caso do tipo apresentado na figura 5.3 é possível de verificar que existe um *Varray* que é o NGIN_APP_COMPILED_RULE_VARRAY, que tem uma posição e outro NGIN_APP_RULE_VARRAY com duas posições. A estrutura XML também permite facilmente verificar pela sua organização de *tags,* a que tipo de dados pertence um subtipo de dados.

5.3.Run TestBeds – O iniciar da ferramenta

Como foi apresentado no subcapítulo 5.1, a ferramenta *TestBeds* é composta por três grandes componentes. O cenário de configuração, o plano de testes e os relatórios gerados do teste.

As principais características da ferramenta são a sua automatização, repetibilidade, transparência e serem genéricas, de forma a não haver diferenças de comportamento nos testes. Para a *TestBeds* testar a rotina X do módulo A ou a rotina Y do módulo B será igual, e o seu funcionamento interno será exactamente o mesmo em qualquer uma das situações. É essencial para o funcionamento das *Testbeds* a sua bateria de testes, pois o armazenamento dos testes possibilitará de forma simples e automática a repetição dos mesmos a qualquer altura, verificando até em alguns casos a evolução ou não, da qualidade das rotinas testadas. A ferramenta tem a capacidade de automatizar os testes, de forma a permitir repetições, ou seja, essa informação de *input* é armazenada de forma a poder ser utilizável sempre que seja necessário, por exemplo uma condição é adicionada à rotina*,* é importante saber se o comportamento da rotina será o mesmo, não só em relação ao resultado esperado, mas também saber se o comportamento é o mesmo de forma a ser retrocompativel com as versões anteriores de *software* e se será igual ou melhor ao que acontecia anteriormente em termos de desempenho, daí ser essencial e imprescindível guardar esta informação sobre os cenários que vão ser usados para "alimentar" as *TestBeds*. Este ponto da retrocompatibilidade tem especial importância pois serão armazenados cenários de teste que poderão ser usados a qualquer momento de forma automática. Os *testSets* serão constituídos por vários casos de teste de forma a abranger o maior número de cenários possíveis de acontecer, aproximando-se assim ao máximo de um cenário real, esta funcionalidade de retrocompatibilidade é o que permite classificar esta ferramenta como capaz de efectuar testes de regressão.

O funcionamento da plataforma de testes apoia-se na informação de configuração que deve ser a mais próxima do real possível e que abranja todas as funcionalidades existentes. Esta "bateria de dados de cenários de teste" é simples, de forma a ser facilmente adicionadas novas funcionalidades ou casos de teste que ainda não estão a ser previstos. Esta bateria de testes será sempre actualizável, e poderá estar sempre a crescer, de forma a tornar-se o mais completo possível, sendo o cenário ideal uma réplica do funcionamento real da solução.

Como apresentado acima a plataforma será composta por sequências de testes (testes compostos) e testes unitários. Numa primeira fase serão elaborados testes compostos que têm como objectivo invocar vários procedimentos, funções ou métodos num único teste. Estes testes compostos incluem *checkpoints* e são denominados de *TestSequences.* Os *checkpoints* serão usados como *flags* aquando da execução de testes, para posteriormente serem comparados, os resultados obtidos com os *outputs* esperados de cada *checkpoint*, sendo estes resultados armazenados para cada *TestSequence*. Será disponibilizado um relatório ao nível de cada *checkpoint*, o que possibilitará um relatório (OK, NOK) de cada invocação, e no caso de NOK, qual o parâmetro que teve resultado diferente do esperado.

A nível estrutural a ferramenta T*estbeds* tem uma hierarquia, o *testset* que é composto por um ou mais testes, que por sua vez podem ter uma ou mais invocações. Se um teste tiver apenas uma invocação estaremos perante um teste unitário, no caso de ter várias invocações estaremos perante uma *TestSequence.*

No final do teste os resultados serão comparados entre os resultados esperados e os resultados obtidos, em cada *checkpoint* da invocação, se tudo correr como o desejado os valores serão iguais e a nova versão do *software* é validada. No caso de serem testes unitários, a plataforma terá um

comportamento interno é exactamente igual. Os relatórios gerados pela ferramenta têm três níveis de detalhe:

- Relatórios ao nível do plano de testes;
- Relatórios ao nível de uma *testSequences* / teste unitário;
- Relatórios ao nível de cada invocação.

Os relatórios gerados tem informações sobre o cenário a ser testado, a data de quando foi efectuado, o tempo de duração do teste e o resultado ou erro devolvido pelo teste efectuado. A estrutura do relatório será apresentada e detalhada mais à frente na figura 5.7.

Com a introdução da *TestBeds* é possível ter um histórico de testes, um quadro evolutivo e um quadro de avaliação de desempenho de cada módulo sendo facilmente testável uma nova versão de um módulo e verificar:

- Se funcionalmente, o resultado foi o esperado ou não, através das *flags* do *checkpoint*;
- Se o resultado é o mesmo das versões anteriores para o mesmo plano de testes.

A *TestBeds* tem no seu procedimento *StartTesting* (*inTestSet, inType*) o seu núcleo, sendo que este será executado pela interface gráfica sendo assim transparente para o utilizador. Este procedimento tem a responsabilidade de gerir automaticamente toda a realização dos testes efectuados. A função vai receber como parâmetros de entrada, o parâmetro *inTestSet* com a informação que diz respeito ao *testset* e o *inType* com a informação referente ao tipo de comportamento que se deseja dar à função *StartTesting*(), visto que este pode ter dois comportamentos, sendo estes:

- Num primeiro caso terá o comportamento mais "normal" e para o qual foi criado, ou seja, um teste é executado e são comparados os seus resultados obtidos com os resultados esperados.
- Num segundo caso, preferencialmente usado antes no cenário apresentado no primeiro caso, será usado para criar os resultados esperados, ou seja, serão feitos testes a rotinas/*packages* que já foram validados como correctos e é assumido pela equipa de testes que têm o comportamento desejado de forma a automaticamente, armazenar os resultados obtidos, na tabela de resultados esperados na bateria de testes da ferramenta.

Não será necessário, a quem vai efectuar o teste alterar ou acrescentar alguma *tag* de PL/SQL. O teste para ser executado, basta ter o modelo de dados da ferramenta preenchido com a informação necessária e tudo configurado de acordo com o teste desejado.

Todos os cenários de configuração vão ser guardados de forma a ser possível no futuro poder executar testes baseados nos mesmos cenários de configuração.

Os testes existentes na bateria de testes podem ser executados até que haja uma alteração ao nível da assinatura de alguma rotina. Enquanto os parâmetros de entrada e saída se mantiverem inalteráveis independentemente da versão do pacote que estejamos a testar, os testes serão realizados, e no caso de se tratar de um teste unitário o resultado obtido terá de ser igual ao testado anteriormente, no entanto se o *testset* for um teste composto (*TestSequences*), o mesmo já não é obrigatoriamente verdade, pois por

causa de invocações anteriores, o comportamento da rotina apesar de ser o mesmo, os *inputs* diferentes levarão a resultados obtidos diferentes do anteriormente testado.

5.4. Funcionamento da ferramenta TestBeds

A ferramenta *TestBeds* tem um funcionamento simples, o utilizador da ferramenta terá de desenhar o plano de testes que deseja, e depois executá-lo. O modelo de dados como foi apresentado anteriormente está dividido em três partes, parte do plano de testes, parte do produto a ser testado e os resultados dos testes.

A primeira fase a efectuar é o carregamento da informação do produto que se vai testar. Depois deste carregamento deverão ser inseridos as configurações que se desejam estar associadas ao produto, e a última fase passa pelo desenho propriamente dito dos testes. Esta fase implica criar um plano de testes, que tem agrupado a si os testes, esses testes estão associados a uma configuração. Depois de criados os testes, são a si associados as invocações (rotinas) que vão ser testadas, esta explicação é facilmente percebida consultando o modelo de dados que se encontra no subcapítulo 5.1.

5.5. Características da ferramenta TestBeds

As principais características desta aplicação centram-se na sua capacidade de:

- **Validar os *packages* PL/SQL desenvolvidos** – a ferramenta foi desenvolvida com o propósito de testar a suite genérica e especificamente os módulos que a constituem;
- **Compatibilidade com tipos complexos** – uma das grandes motivações para a opção em desenvolver uma ferramenta de raiz, foi a limitação que existia nas ferramentas comerciais, pois estas não possibilitavam de uma forma simples e clara a manipulação de tipos de dados não nativos. A ferramenta através das funções de serialização e deserialização permite a manipulação desses tipos de dados;
- **Mecanismo de *"Golden Run"* para inserção de resultados esperados** – A inserção de todos os resultados esperados, num plano de testes complexo pode ser algo moroso. Uma forma de optimizar essa inserção passa por este mecanismo de *"golden run"*, que o que faz é executar o teste modificando apenas a sua essência primária. Em vez de comparar os resultados obtidos com os resultados esperados, a ferramenta insere os resultados obtidos do teste, na tabela de resultados esperados para o *testSet.* Isto é bastante útil em casos em que exista uma versão do produto que está com o funcionamento correcto e pode ser usada como ponto de partida para testar versões que sejam lançadas a seguir.
- **Possibilidade de declarar os *inputs* e *outputs* esperados manualmente** – A definição manual dos parâmetros esperados permite alterar algum ou todos os resultados de uma invocação.
- ***Outputs* podem não sofrer avaliação** – Caso existam parâmetros de *output* que não se pretenda avaliar, a ferramenta permite definir quais os resultados obtidos que o utilizador pretende avaliar e os que não pretende, por exemplo, uma variável que seja preenchida com a data actual do teste pode ser interessante em alguns casos que não seja avaliado de forma a não tornar o teste inválido.
- **Realimentação de parâmetros (*inputs* de uma invocação podem ser alimentados por *outputs* de invocações anteriores)** – esta característica é bastante útil em testes que tenham dependências entre invocações. Se definirmos esta dependência quando a invocação de origem é executada e o resultado obtido é guardado, os parâmetros de entrada das invocações seguintes podem herdar esse valor e usá-lo no seu próprio teste. Imaginemos um teste cuja primeira invocação retornará o número de um cliente mediante um conjunto de características, esse número depois poderá ser usado como parâmetro de entrada de outra invocação.

- **Avaliação de retrocompatibilidade entre versões** – Como a ferramenta permite armazenar as baterias de casos de teste, de produtos e configurações, quando um novo produto é disponibilizado, facilmente é possível voltar a executar o conjunto de testes mas agora sobre o novo produto, existindo ainda a funcionalidade de comparar os resultados obtidos, com os resultados obtidos da versão anterior.
- **Suporte ao polimorfismo de funções** – Na ferramenta é possível usar rotinas que tenham o mesmo nome mas que tenham assinaturas diferentes. Muitas vezes quando existem evoluções no produto, novos parâmetros são acrescentados. A ferramenta permite importar as várias rotinas existentes mesmo que tenham o nome igual.
- **Cópia de testes anteriormente configurados, integral ou por referência entre diferentes produtos ou no mesmo produto –** É possível fazer a cópia integral ou por referência de testes. Se no primeiro caso o que a aplicação faz é a duplicação dos testes, no segundo caso o *testSet* fica referenciado com outro teste, sendo possível a referência de todos os testes ou apenas de alguns testes desse plano de testes.
- **Cópia dos parâmetros de configuração entre diferentes produtos ou no mesmo produto -** É possível copiar parâmetros de configuração definidos num produto para outro, por exemplo, uma data pode ser definida num produto, sendo que essa variável pode ser copiada para outro produto permitindo assim reduzir o tempo de configuração do teste.
- **Relatório dos testes efectuados –** A ferramenta gera três tipos de relatórios. Relatórios que retratam todo o conjunto de testes, relatórios de cada teste e relatórios de cada invocação, sendo que o relatório do conjunto de testes apresenta informação sobre o conjunto de testes a executar, bem como os testes que o constituem e as respectivas invocações.
- **Histórico de testes** – O armazenamento dos relatórios permite ter um histórico de testes.

5.6.Apresentação da ferramenta TestBeds

O aspecto geral da aplicação *TestBeds* é ilustrado na figura 5.4.

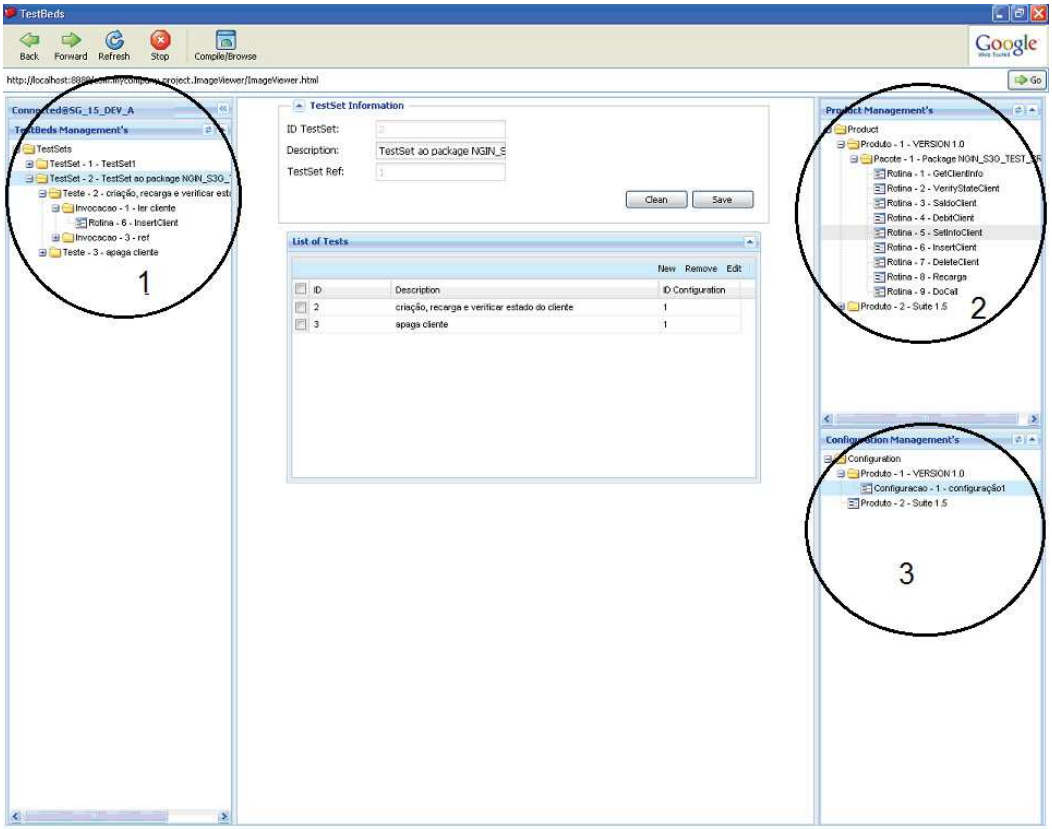

Figura 5.4 - *Layout* inicial das TestBeds

Pode ver-se que a janela inicial da ferramenta divide-se em três árvores de configuração e numa área principal onde são visualizados os detalhes do item seleccionado. As árvores de configuração são:

- ***TestBeds Management's***: árvore do plano de testes desenhado (marcado com 1 na figura 5.4).
- ***Product Management's:*** produtos existentes e que podem ser testados (marcado com 2 na figura 5.4).
- ***Configuration Management's:*** configuração sobre as quais os produtos vão ser testados (marcado com 3 na figura 5.4).

5.7. Requisitos da ferramenta

Os principais requisitos da ferramenta encontram-se sintetizados de seguida:

- Acesso à aplicação através de uma interface *web*. A aplicação corre num *Tomcat*.
- Ter *grants*[22] de todos os objectos do *schema*[23] testado e aos respectivos *schemas* Data e auxiliares ao utilizador.
- Na primeira vez que se utiliza a ferramenta num *schema* novo de testes deve escolher-se a opção *New Connection* e completar os campos *Name*, *Version*, *Username* e *Password*. Esta informação será usada para futuras autenticações.

5.8. Execução da TestBeds

Para a execução de um *TestSet* e os seus respectivos *TestSets* de referência (caso existam) deve-se clicar na árvore do plano de testes sobre o nó do *TestSet* desejado. O utilizador deve escolher no menu a opção *Execute TestSet* e a seguir escolher uma entre as duas soluções disponíveis e apresentados na figura 5.5:

- Solução ***'CheckPoint'***: Os valores resultantes da invocação da rotina são comparados aos valores esperados.
- Solução ***'GoldenRun'***: Inserem-se os *outputs* esperados a partir de uma versão de *software* que esteja já testada, aceite e pronta a ser entregue para o cliente. Reduz-se assim o tempo necessário na inserção dos resultados esperados. Na solução ***'GoldenRun'*** nenhum relatório é criado.

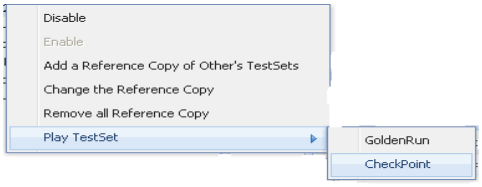

Figura 5.5 – Execução de um *TestSet*

[22] Concede privilégios específicos para um objecto (tabela, vista, sequência, função) para um ou mais utilizadores ou grupos de utilizadores [51].

[23] É essencialmente um espaço lógico numa Base de dados que permite definir objectos (tabelas, tipos de dados, funções e operadores), cujos nomes podem ser iguais a outros objectos existentes noutros esquemas [52].

Ou seja, num cenário ideal primeiramente corre-se o teste com a solução ***GoldenRun*** sobre uma versão que esteja correcta e sem erros servindo esta, para definir os valores esperados e a seguir para uma nova evolução de versão de *software* corre-se o teste com a solução ***CheckPoint***. Após a execução:

- Na solução ***'CheckPoint'***: são gerados relatórios dos testes efectuados. Há a comparação dos parâmetros de *output* da invocação com os *outputs* esperados, **OK, NOK**.
- Solução ***'GoldenRun'***: resultados são obtidos e visualizados através da janela da lista dos parâmetros.

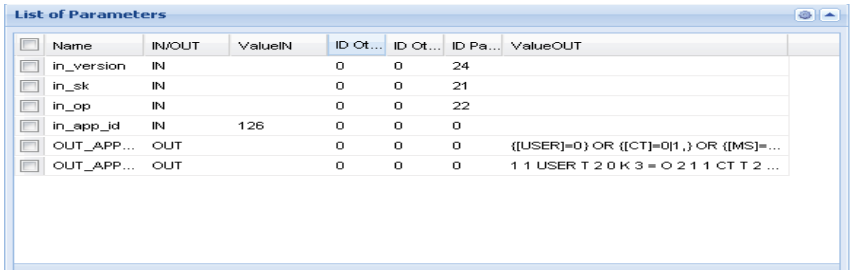

Figura 5.6 – Apresentação dos resultados após *"Golden Run"*

5.8.1. Relatório final gerado

Os relatórios resultam da comparação entre os *outputs* obtidos e os esperados. Após a execução com a solução ***'CheckPoint'*** surge a janela com os relatórios gerados. Um exemplo é o apresentado na figura 5.7:

```
Date: 12-06-2010 11:30:25
Suite Version: 1.4
Test Set ID:2        Description: Conjunto de testes ao Status
Config ID:  1        Description: (version=1;sk=200;operator=0;user='TESTBED';msg='TESTE')
Test ID:    1        Description: Teste ao Status
Test ID:    5        Description: Teste ao CNS

**************************************************************************
1 --- Teste ao Status
**************************************************************************

RESULTS
--------------------------------------------------------------------------------------------------------------
Rotina                  Rotina ID    Description                                                       Result    Execution Ti
geteventid              1            getEventId: Evento desconhecido                                   NOK       5.674
--------------------------------------------------------------------------------------------------------------
-Parametro errado - outerror_abs - ID = 9
--valor esperado=25131
--valor obtido=2513

geteventid              1            getEventId: Evento conhecido (sucesso)                            OK        4.496
--------------------------------------------------------------------------------------------------------------
get_nextstatus          2            get_nextStatus: Salto do pre-activo para o activo com evento CALL OK        28.57
--------------------------------------------------------------------------------------------------------------
get_nextstatus          2            get_nextStatus: Sem proximo estado por tempo                      OK        8.109
--------------------------------------------------------------------------------------------------------------
get_nextstatus          2            get_nextStatus: Sem proximo estado por tempo, mas NS e NSD passada por per   OK   6.947
                                     ametro
--------------------------------------------------------------------------------------------------------------
RESULTS TEST--- NOK.
**************************************************************************

**************************************************************************
5 --- Teste ao CNS
**************************************************************************

RESULTS
--------------------------------------------------------------------------------------------------------------
Rotina                  Rotina ID    Description                                                       Result    Execution Ti
get_nextstatus          2            get_nextStatus: Salto do pre-activo para o barrado por Tempo      OK        11.254
--------------------------------------------------------------------------------------------------------------
get_nextstatus          2            get_nextStatus: entity a NULL                                     OK        5.778
--------------------------------------------------------------------------------------------------------------
delete_non_persistent   20           delete_consumptions: Limpar os consumos do cliente                OK        8.355
--------------------------------------------------------------------------------------------------------------
process_cns             9            Um process_cns com todos as periodicidades, com e sem periodic_ini, com pe   OK   150.355
                                     riodicqt a 1 e a 2 para cada periodicidade
--------------------------------------------------------------------------------------------------------------
get_consumptions        10           get_consumptions: Devolve todos os consumos                       OK        33.395
--------------------------------------------------------------------------------------------------------------
deactivate_cns          11           deactivate_cns em todos os cons_types                             OK        5.296
--------------------------------------------------------------------------------------------------------------
get_consumptions        10           get_consumptions: Devolve todos os consumos                       OK        28.71
--------------------------------------------------------------------------------------------------------------
process_cns             9            process_cns: Process_cns depois de um deactivate_cns              OK        133.445
--------------------------------------------------------------------------------------------------------------
get_consumptions        10           get_consumptions: devolve todos os consumos                       OK        51.904
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
RESULTS TEST--- OK.
**************************************************************************

RESULTS TESTSET
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
NR. Invocações OK=13
NR. Invocações NOK=1
FINAL RESULT: NOK
FINAL TIME: 9.32961 (s)
```

Figura 5.7 - Exemplo de relatório detalhado

Observa-se na figura 5.7 que existe referência aos detalhes do plano de teste. Data e versão são listados de forma a identificar claramente o relatório. São apresentados os testes que foram efectuados, a descrição das rotinas que o compõem e os resultados após o *checkpoint* (**OK/NOK**). É apresentado o resultado geral do teste tendo em conta os resultados obtidos nas invocações das rotinas. Por fim é apresentado o resultado final do *Testset*. Observa-se que existe sempre a avaliação do *testSet*, do teste e da invocação.

Se existir uma invocação com **NOK** é apresentado qual o parâmetro em que o erro aconteceu qual o valor obtido e qual o valor esperado.

No anexo 1 é apresentado um cenário de utilização da ferramenta TestBeds.

6. Descrição do trabalho no âmbito do projecto *Configuration Manager*

O objectivo deste sexto capítulo, é apresentar ao leitor a segunda ferramenta desenvolvida, o *Configuration Manager*. É apresentada uma visão geral sobre a ferramenta bem como a sua arquitectura. Para finalizar são apresentados os aspectos mais relevantes e inovadores da ferramenta.

O CM é composto por diversos módulos que cooperam entre si com o objectivo de definir entidades e regras que determinam o comportamento do sistema NGIN. Esse conjunto e combinação de entidades e regras dão origem a produtos/ofertas comercias diferenciadas que constituíram o catálogo de produtos/serviços que as operadoras vão possuir. Para o seu âmbito de configuração, dos subsistemas CARE e CORE, são disponibilizados mecanismos de:

- Abstracção de conceitos de negócio;
- Minimização de erros no processo de configuração com recurso a *wizards* que conduzem o utilizador no processo de configuração de uma oferta comercial ou entidade;
- Agilização do processo de edição da configuração com recurso a mecanismos de edição em massa;
- Possibilidade de consulta da configuração das ofertas comerciais, bem como outras entidades com mecanismos de exploração de relações entre entidades de configuração;
- Gestão da topologia e modelos activos;
- Extensão à configuração de outros sistemas.

Fazendo uma breve cronologia sobre a forma de configuração da plataforma NGIN:

- **Até 2004** - a configuração era manual directamente no modelo de dados.
- **De 2004 até 2009** – é usada a ferramenta PCA;
- **De 2009 até agora** – surge a ferramenta CM;

A ferramenta (PCA) de configuração dos sistemas NGIN quando surgiu foi uma grande evolução. No entanto, a sua aceitação por parte dos clientes foi sempre complicada, pois é necessário, a quem configura, possuir um conhecimento aprofundado do negócio, a configuração não é orientada, tornando-a bastante difícil. Dada a fraca aceitação desta ferramenta, muitas das configurações dos clientes são asseguradas pela própria PT Inovação, o que obriga a ter recursos permanentemente alocados com estas tarefas, quando deveria ser o próprio cliente a efectuá-las. Devido a esse facto, surge a necessidade de criar uma ferramenta mais amigável para o utilizador e que seja capaz de abstrair ao máximo o conhecimento do negócio a quem configura, tornando muito mais fáceis as configurações dos sistemas NGIN, permitindo aos clientes efectuar as suas próprias configurações. Aliado a este factor a evolução da solução NGIN, que a ser composta pelos subsistemas NGIN Care e NGIN Core, tornou a ferramenta PCA obsoleta pois esta apenas configura o subsistema NGIN Core.

6.1. Visão geral

Para se perceber melhor o que existia antes do CM e o que foi proposto fazer, na figura 6.1 é possível observar uma visão geral da arquitectura da ferramenta de configuração PCA e do CM. Na figura o CM está representado pelo módulo *Wizard Based* enquanto o PCA é o módulo *GUI expert/Custom GUI* que liga directamente ao modelo passivo das plataformas.

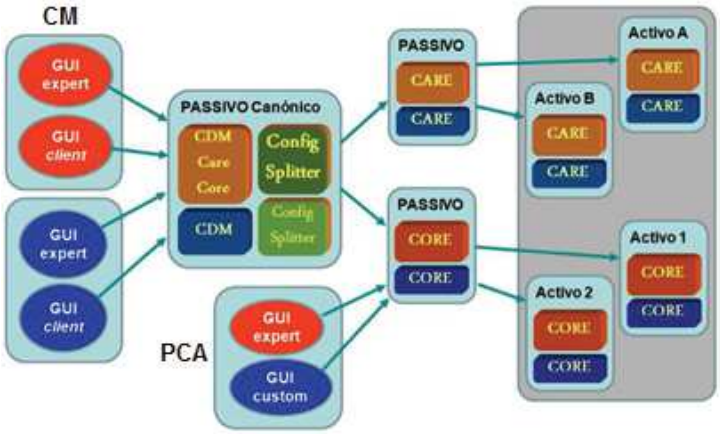

Figura 6.1 – Arquitectura PCA vs CM

A principal novidade que existe no desenho da nova arquitectura para o CM foi a incorporação do modelo canónico[24] com o objectivo de abstrair as configurações das plataformas NGIN Care e NGIN Core. É sobre este modelo que todas as configurações irão ser feitas, ao contrário do que acontece no PCA que a ferramenta configura directamente sobre uma cópia local da plataforma NGIN Core. Conforme foi referido, é o CM que trata todos os aspectos de configuração, estando fora do seu âmbito aspectos como o aprovisionamento. Conceptualmente, lida com as configurações de negócio, gestão do ciclo de vida de versões e com mecanismos de distribuição e activação dessas versões.

Para se perceber melhor a visão conceptual do CM é apresentada na figura 6.2.

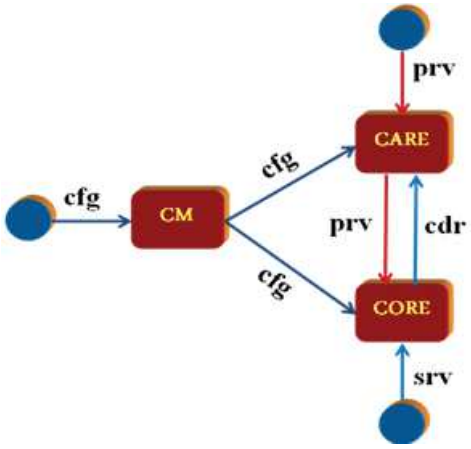

Figura 6.2 – Visão Conceptual do *Configuration Manager*

[24] Modelo unificado de suporte ao CM que abstrai os modelos activos dos sistemas a configurar garantindo que a abstracção é feita numa camada inferior ao ambiente gráfico. É um modelo orientado à entidade e não ao módulo

6.2. Arquitectura do Configuration Manager

Para dar resposta às novas necessidades, a arquitectura que existia anteriormente, da ferramenta de configuração PCA, foi estendida para albergar os novos componentes (CDM e Splitter) e os novos modelos de negócio (NGIN CARE passivo e NGIN CARE activo).

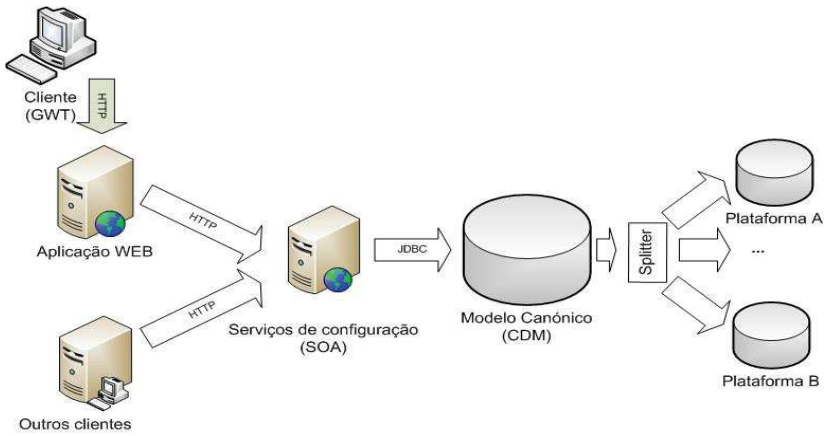

Figura 6.3 – Arquitectura SOA no CM

Conforme se pode verificar pela figura apresentada acima, o CM possui uma arquitectura orientada ao serviço (SOA[25]). A camada de serviços de configuração pode ser consumida através de invocações SOAP[26] (*Simple Object Access Protocol*). Todos os serviços expostos nesta interface são autónomos e *stateless*, ou seja, a invocação de um serviço não depende de invocações prévias nem de contexto de sessões de configuração.

O *Configuration Manager* está bem definido quanto a sua estrutura. Está dividido em módulos sendo que cada módulo tem a sua função bem definida.

O GUI (*Graphical User Interface*), é um módulo composto pelo ambiente gráfico de configuração onde é possível proceder à gestão de versões, gestão de topologia e configuração do negócio da solução (configuração de ofertas comerciais e entidades que as constituem), segue o padrão MVC[27] (*Model-View-Controller*).

O módulo GWT *(Google Web Toolkit) Services,* disponibiliza as funcionalidades necessárias ao GUI (*Fat Client*), sendo todos os serviços deste módulo orientados às necessidades da interface gráfica. Estes dois módulos são orientados ao interface gráfico.

[25] *Service-Oriented Architecture* é um estilo de arquitectura de *software* cujo princípio fundamental defende que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços. Disponibiliza interfaces acessíveis através de *web services* ou outra forma de comunicação entre aplicações. A arquitectura SOA é baseada nos princípios da computação distribuída e utiliza o paradigma pedido/resposta para estabelecer a comunicação entre os sistemas clientes e os sistemas que implementam os serviços [53].

[26] *Simple Object Access Protocol* é um protocolo para troca de informações estruturadas numa plataforma descentralizada e distribuída. Baseia-se em XML para o formato das suas mensagens, e normalmente usa outros protocolos como o RPC e HTTP, para comunicação e transmissão de mensagens [54].

[27] MVC é um padrão de arquitectura de *software* que visa separar a lógica de negócio da lógica da interface, permitindo o desenvolvimento, teste e manutenção isolado de ambos [55].

O *Basic Services* é um módulo que disponibiliza as funcionalidades de edição da configuração e gestão do seu ciclo de vida. Os serviços que disponibilizam são orientados ao negócio/configuração. Estes serviços podem ser acedidos via APIs SOAP.

A LCV *(Lógica de controlo de versões),* é um módulo que garante a gestão da topologia, a gestão do ciclo de vida das versões de configuração e controlo das fases de *splitting*, transferência e activação de configurações.

O modelo canónico (CDM), é um modelo unificado de suporte ao CM que abstrai os modelos activos dos sistemas a configurar garantindo que a abstracção é feita numa camada inferior ao ambiente gráfico. É um modelo orientado à entidade e não ao módulo.

O módulo SPLITTER, é o responsável pelo mecanismo de transformação e separação das configurações entre o modelo canónico (CDM) e os modelos passivos, recorrendo para isso a regras de *splitting* onde se encontra depositada toda a lógica de transformação de conceitos genéricos em elementos específicos de cada subsistema.

O modelo passivo é a representação local dos modelos de dados de configuração dos sistemas a configurar.

O modelo activo, diz respeito aos modelos de dados "reais" de configuração dos sistemas a configurar. A figura 6.4 espelha esta divisão entre os diferentes módulos.

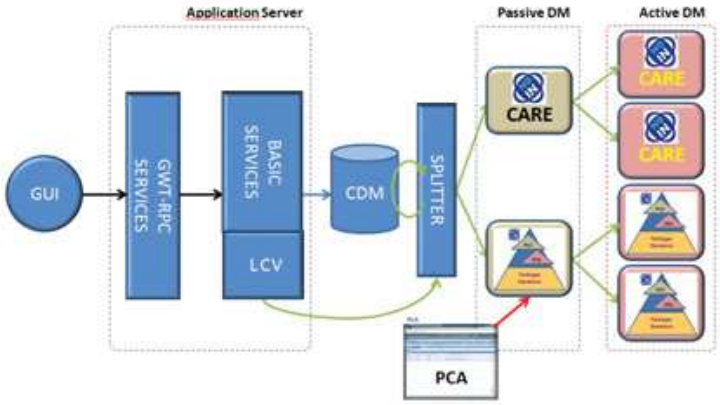

Figura 6.4 - Visão geral da arquitectura CM vs PCA

O CM caracteriza-se, fundamentalmente, pela agilização de todo o processo de configuração, usando para isso um ambiente gráfico guiado e orientado a conceitos perceptíveis pelo utilizador.

6.2.1. Integração do GWT com *Spring*

Apesar das "maravilhas" que o GWT consegue fazer com a comunicação cliente-servidor e na transparência dessa interacção para quem desenvolve os serviços, estes não são *web services*[28]. Na

[28] Web service é uma tecnologia baseada em XML e HTTP cuja principal função é disponibilizar serviços interactivos na Web que podem ser acedidos (ou consumidos) por qualquer outra aplicação independente da linguagem ou plataforma em que a aplicação foi construída [56].

verdade estes são apenas *servlets* que conseguem ”serializar/deserializar” a informação transmitida. O facto de se ter que estender uma classe (*RemoteServiceServlet*) levanta algumas questões como:

• Se quisermos aproveitar alguma lógica já implementada anteriormente com a sua própria hierarquia?

• Se quisermos aproveitar essa mesma lógica para disponibilizar o serviço através de *WebServices SOAP* ou *RESTful*?

• Se quiser usar um serviço interno e não necessitar de toda a parte dos RPC’s?

• E se entretanto se mudar de tecnologia (GWT) e só se quiser substituir a camada de apresentação?

De modo a endereçar todas estas questões foi usado o GWT integrado com o *Spring MVC* que possui uma implementação de injecção de dependências [29](*Dependency Injection*), tornando o código mais legível e deixando a tarefa de inicialização ou de configuração para o *container*, neste caso o *Spring*, tornando-se assim num *standard* da indústria. Sem entrar em muitos detalhes o *Spring* foi desenvolvido com o objectivo de tornar o código o mais separado possível das dependências promovendo o desenvolvimento baseado em *POJO’s*[30] *(Plain Old Java Objects)* para aplicações empresariais. A figura 6.5 apresenta a implementação de um serviço RPC com integração do Spring. [9]

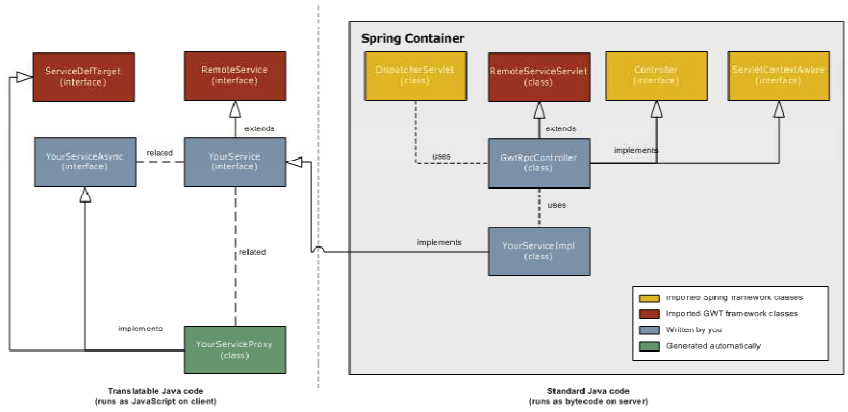

Figura 6.5 - Implementação de um serviço RPC com integração do *Spring* [9]

Esta nova abordagem de implementação dos Serviços-GWT vai permitir uma separação da implementação ”real” dos serviços, da camada de apresentação (GWT). Na realidade, os Serviços-GWT não irão possuir lógica nenhuma, a sua tarefa será delegar as invocações para a implementação nos serviços base.

[29] Injecção de dependências é um padrão de desenvolvimento programação que visa desacoplar os componentes de uma aplicação. Nesta solução as dependências entre os módulos não são definidas programaticamente, mas sim pela configuração de uma infra-estrutura de *software* (*container*) que é responsável por "injectar" em cada componente as dependências definidas [57].

[30] *Plain Old Java Objects* é um objecto *Java* que não precisa de implementar nenhuma interface ou estender de nenhuma classe por obrigação de um *framework* ou *container* [58].

6.2.2. Cliente GWT

A estrutura principal do CM é controlada pela implementação MVC do GXT (Ext GWT, extensão do GWT usada neste projecto). Funciona da seguinte forma: o *dispatcher* lança um evento para todos os controladores. Se o controlador estiver à escuta desse evento, o evento é processado. Caso contrário não faz nada com esse evento. O controlador é responsável pela lógica e por actualizar os modelos. As vistas, que estão ligadas aos controladores, são actualizadas após um modelo ser actualizado. Por fim, para tudo isto acontecer o *dispatcher* terá que registar todos os controladores para os quais terá que enviar os eventos. Para se perceber melhor a implementação do padrão MVC do GXT vejamos o seguinte exemplo. Inicialmente são registados todos os controladores no *dispatcher* (figura 6.6).

```
public class CM implements EntryPoint {
. . .
public void onModuleLoad() {
. . .
// Registar todos os controladores no dispatcher
Dispatcher dispatcher = Dispatcher get ();
dispatcher.addController (new AppController ());
dispatcher.addController (new MainPlafondController ());
dispatcher.addController (new MainLifeCycleController ());
dispatcher.addController (new NavigatorController()) ;
dispatcher.addController (new WizardController ());
dispatcher.addController (new VersionsController ());
. . .
}
}
```

Figura 6.6 - Registo dos controladores no *Dispatcher*

Neste momento temos os controladores registados no *dispatcher*. A partir daqui é possível lançar eventos para estes controladores através do *dispatcher*. Existe um *enum* onde todos os eventos da aplicação estão enumerados (figura 6.7).

```
public enum AppEvents {
. . .
INIT(10000 , ”INIT” ) ,
VERSIONING SELECTED(2 , ”VERSIONING SELECTED” ) ,
LOGOUT SELECTED(4 , ”LOGOUT SELECTED” ) ,
. . .
private final int evtType;
private final String evtDesc;
AppEvents (int eventType , String eventDescription){
this.evtType = eventType ;
this.evtDesc = eventDescription;
}
public final int getEventType() {
return evtType ;
}
public String getEventDescription () {
return evtDesc ;
}
public static AppEvents getAppEventById ( int eventType ) {
for (AppEvents evt : AppEvents . values ( )) {
i f ( evt.getEventType ( ) == eventType ) {
return evt ;
}
}
return null;
}}
```

Figura 6.7 - *Enum* com os eventos da aplicação

A partir deste momento, de qualquer ponto da aplicação é possível lançar eventos para os controladores.

```
. . .
dispatcher.dispatch(AppEvents.INIT.getEventType()
);
. . .
```

Figura 6.8 - Lançamento de eventos

No caso aqui descrito na figura 6.8 é lançado o evento INIT que notifica todos os controladores que estão à escuta deste evento que a aplicação iniciou. Apenas os controladores que estão à escuta deste evento o irão processar.

```
public class AppController extends Controller{
private AppView appView ;
public AppController(){
registerEventTypes (AppEvents.INIT.getEventType
());
. . .}
```

Figura 6.9 - Registo dos eventos nos controladores

A figura 6.9 apresenta o registo de eventos, neste caso na classe AppController estando esta à escuta do evento INIT. Quando o evento INIT é recebido no controlador este é processado e enviado para a respectiva vista, como é demonstrado na figura 6.10.

```
public class AppController extends Controller{
. . .
public void handleEvent (AppEvent event) {
switch
(AppEvents.getAppEventById(event.type)){
case INIT:
on Init(event);
break;
. . .
}
}
. . .
private void on Init (AppEvent event){
forwardToView(appView,event);
}
. . .
```

Figura 6.10 - Processamento dos eventos nos controladores

Para este caso e como se trata do evento que inicia toda a aplicação, a vista vai tratar de ”renderizar”[31] o interface gráfico da aplicação como demonstra a figura 6.11.

[31] Este termo é usado quando o *browser* já recebeu uma nova página (*script html*, por exemplo), mas o ecrã ainda não está totalmente desenhada.

```
public class AppView extends View {
. . .
public AppView(Controller controller){
super (controller);
}
. . .
protected void handleEvent (AppEvent event){
switch (AppEvents.getAppEventById (event.type)){
case INIT:
on Init (event);
break;
. . .
}
}
private void initUI(){
viewport=new Viewport();
viewport.setLayout(new BorderLayout());
createHeader();
createCenter();
createFooter();
Registry.register(”viewport”,viewport);
Registry.register(”center”,center);
Registry.register(”header”,header);
Registry.register(”footer”,footer);
RootPanel.get().add (viewport);
}
. . .
}
```

Figura 6.11 - Envio de evento para a vista

Na figura 6.12 encontra-se o esquema MVC presente na aplicação. A acrescentar ao MVC tradicional, existe mais um módulo que é os Serviços GWT-RPC. O controlador é o responsável por fazer as invocações a estes serviços que se encontram do lado do servidor.

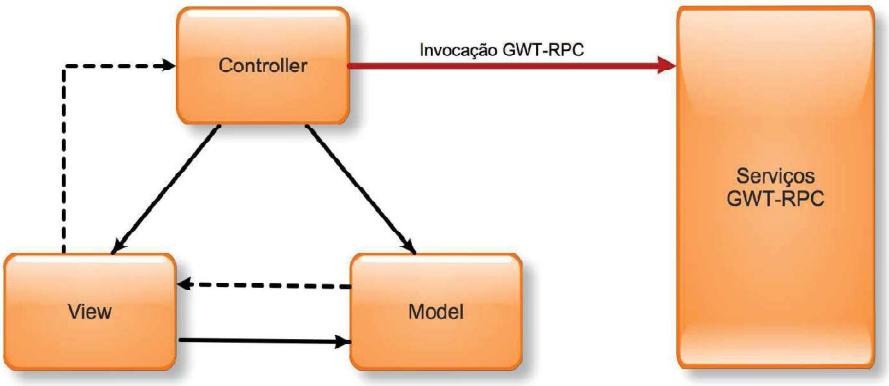

Figura 6.12 - Padrão MVC

6.2.3. Serviços GWT-RPC

Os serviços GWT-RPC consistem num conjunto de operações que disponibilizam todas as funcionalidades necessárias ao funcionamento da componente de interface gráfica do CM.

Estes serviços retornam objectos com a representação da informação que será apresentada ao cliente. Conforme o ilustrado na figura 6.12 apresentada atrás, a invocação destes serviços será efectuada pelos controladores existentes do lado do cliente. Após a invocação do serviço, o modelo é actualizado do lado do cliente, o qual irá notificar a vista apresentada ao cliente para que passe a representar a nova informação retornada.

Os serviços disponibilizados nesta camada podem ser divididos em três categorias:

- Serviços orientados à escrita, leitura e manipulação de versões de configuração.
- Serviços de consulta e manipulação de informação respeitante a um utilizador em particular. Estes serviços recorrem a um modelo de apoio para armazenar informação do cliente.
- Serviços para obtenção de informação relativa a controlo de acessos.

6.2.4. Serviços Base

Como ilustra a figura 6.13 é importante disponibilizar estes serviços como forma de facilitar o processo de integração (preferencialmente *standard*), uma vez que existe a necessidade de consulta de informação de configuração por outras entidades. As ferramentas de suporte ao negócio são um exemplo típico deste tipo de necessidade. Após extraírem a informação dos modelos activos para posterior inserção na *Data Warehouse*, necessitam de consultar o repositório de configuração para obter informação da configuração que não foi transportada para os modelos activos. As descrições são o caso mais comum.

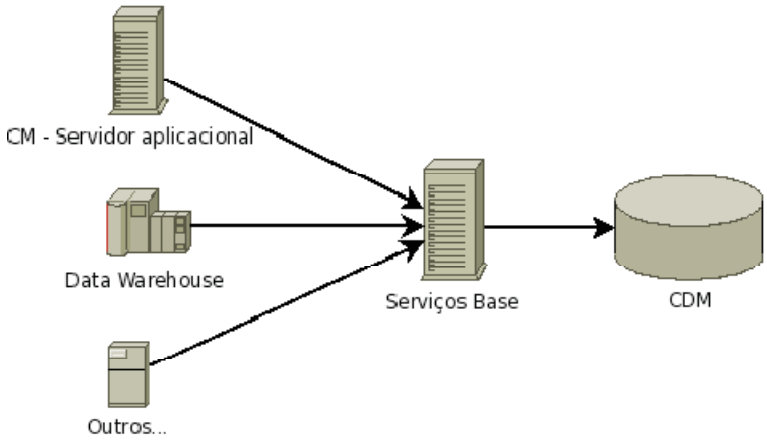

Figura 6.13 – Serviços Base

Adicionalmente, a identificação destes serviços potencia a reutilização e a redução de um sistema complexo em partes mais pequenas e autónomas, levando, consequentemente, à simplificação do problema, tornando a evolução e manutenção um processo mais fácil. O desenvolvimento destes serviços seguiu uma abordagem, que é apresentada de seguida. De forma a se perceber melhor a abordagem, são expostos alguns exemplos da entidade *plafond*, omitindo-se no entanto alguns detalhes não relevantes. Após a identificação das entidades de configuração, estas são descritas num documento (*XSD* - XML Schema Definition).

O processo de definição das entidades de configuração num *schema* é trabalhoso, no entanto, permite que a sua definição seja efectuada num formato *standard*, aumentando a interoperabilidade bem como, facilitando o processo de desenvolvimento de outros requisitos existentes na aplicação (por exemplo, comparação entre duas entidades de configuração do mesmo tipo, neste caso, comparando duas configurações de *plafond's* distintas). Na figura 6.14 é apresentada a definição, embora parcial, da entidade de configuração *plafond*. Para se perceber melhor o que é um *plafond* neste contexto, este representa o saldo do cliente, ou seja, um valor de uma determinada unidade (normalmente dinheiro) que o cliente dispõe para realizar "eventos".

```
<xsd: element name="plafond">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd : extension base="tns:plafondBase">
        <xsd: sequence>
          <xsd:element name=" saldoInicial" type="xsd:float">
            <xsd:annotation>
              <xsd:documentation>
                Representa o saldo do cliente no momento de activacao do servico.
              </ xsd:documentation>
            </ xsd:annotation>
          </ xsd:element>
. . .
</xsd: element>
```

Figura 6.14 - Definição da entidade de configuração plafond

Após a definição da entidade, passa-se à definição dos pedidos e respostas correspondentes às operações disponibilizadas sobre a referida entidade. Na figura 6.15 está demonstrada a definição de um pedido de criação de um *plafond* e a respectiva resposta.

```
<! Formato de um pedido de criacao de um plafond>
 <xsd:element name=”createPlafondRequest”>
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:extension base=”tns:request”>
     <xsd:sequence>
      <xsd:element ref=”tns:plafond”/>
     </xsd:sequence>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
<! Resposta de um pedido de criacao de um plafond>
<xsd:element name=”createPlafondResponse”>
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base=”tns:response”>
    <xsd:attributeGroup ref=”tns:idsAttr”/>
   </xsd:extension>
  </xsd:complexContent>

 </ xsd:complexType>

</ xsd:element>
```

Figura 6.15 - Definição dos pedidos e respostas das entidades de configuração

Como está ilustrado atrás estes serviços seguem uma abordagem SOA. Para se desacoplar os serviços do cliente foi criado um *jar*[32] comum, tanto aos serviços como ao cliente, que irá conter os DTO [33](*Data Transfer Objects*) e as assinaturas dos serviços. Com base nos *schemas* definidos anteriormente, recorreu-se a uma ferramenta (*Castor – Source Generator*) para gerar os DTO. Após a definição da entidade de configuração e as respectivas operações sobre a mesma, passou-se à definição do interface, conforme se pode verificar na figura 6.16.

```
public interface IPlafond Service{
        CreatePlafondResponse createPlafond(CreatePlafondRequestrequest);
…
}
```

Figura 6.16 - Definição do interface das operações

Após a definição do interface existe a necessidade da implementação. O exemplo apresentado aqui é uma tarefa relativamente simples, pois resume-se à persistência do *plafond* que se pretende criar. Naturalmente que a responsabilidade de persistir a informação é delegada para uma camada de acesso a

[32] *Java Archive* (JAR) é um ficheiro compactado usado para distribuir um conjunto de classes *Java*. É usado para armazenar classes compiladas e metadados associados que podem constituir um programa [59].

[33] *Data Transfer Objects* é um padrão de desenho usado para transferir dados entre subsistemas das aplicações. Os DTOs são com frequência usados em conjunto com *Data Access Objects* para recolher dados de bases de dados relacionais [60].

dados (DAO[34]), de forma a evitar qualquer dependência entre os serviços e o suporte da informação. O passo seguinte é expor os serviços usando o *Spring Remote*. Protocolos disponíveis incluem *RMI, Spring's HTTP Invoker, Hessian, Burlap, JAX-RPC, JAX-WS e JMS*. A parte importante é que o protocolo a usar é configurado na configuração do *Spring* e que as classes Java não possuem nenhum conhecimento do protocolo a usar. No ficheiro *springconfig-services.xml* é definido o *bean* do serviço exposto como ilustra a figura 6.17.

```
<?xml version=" 1.0" encoding="UTF−8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
default−autowire="byName"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring−beans−2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring−context−2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/t/spring−tx−2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring−aop−2.5.xsd">
<context:annotation−config/>
<!−− ==================== SPRING Services==================== −−>
. . .
<bean class="pt.ptinovacao.cm.service.impl.PlafondServiceImpl"
id="plafondService">
</bean>
. . .

</beans>
```

Figura 6.17 - Configuração do bean do serviço

Por fim, para expor o serviço por *Spring's Remoting* e definido no ficheiro *service-servlet.xml* o que está na figura 6.18.

[34] *Data Access Object* é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a bases de dados. Numa aplicação que utilize a arquitectura MVC, todas as funcionalidades da base de dados, tais como obter as conexões, mapear objectos Java para tipos de dados SQL ou executar comandos SQL, devem ser feitas por classes de DAO [61].

```
<?xml version=”1.0”encoding=”UTF−8”?>
<!DOCTYPE beans PUBLIC ”−//SPRING//DTD BEAN//EN”
”http://www.springframework.org/dtd/spring−beans.dtd”>
<beans>
. . .
<!−−+
|//PLAFOND
+−−>
<bean id=”httpExporterPlafondService”
class=”org.springframework.remoting.ht tpinvoker.HttpInvokerServiceExporter”>
<property name=”service”ref=”plafondService”/>
<property name=”serviceInterface”
value=”pt.ptinovacao.cm.service.IPlafondService”/>
</bean>
. . .
<bean id=”urlMapping”
class=”org.springframework.web.servlet.handler.SimpleUrlHandlerMapping”>

<property name=”mappings”>
<props>
<prop key=”/plafondService.httpInvoker”>
httpExporterPlafondService
</prop>
. . .
</props>
</property>
</bean>

</beans>
```

Figura 6.18 - Exposição do serviço por HTTP

Para tratar da persistência dos dados foi usada a ferramenta Ibatis da *Apache Software Foundations*. A camada DAO foi gerada através da utilização de uma outra ferramenta (Ibator), que através da introspecção da base de dados gera esta camada. O Ibator gera:

- Mapeamentos de *queries* em XML;
- Classes Java para mapear os dados da tabelas;
- Classes DAO que usam as classes anteriores.

Com a ajuda do *Spring* é igualmente fácil expor estes serviços por *web services* criando-se os *endpoints* necessários, os quais interceptam um pedido e com base no elemento raiz do documento e o *namespace*, ao qual o elemento pertence, delegam para a implementação do serviço.

```
<?xml version=”1.0”encoding=”UTF−8”?>
<!DOCTYPE beans PUBLIC ”−//SPRING//DTD BEAN//EN”
” http://www.springframework.org/dtd/spring−beans.dtd”>
<beans>
. . .
<!−−+//PLAFOND+−−>
<bean id=”httpExporterPlafondService”
class=”org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter”>
<property name=”service”ref=”plafondService”/>
<property name=”serviceInterface”
value=”pt.ptinovacao.cm.service.IPlafondService”/>
</bean>
. . .
<bean id=”urlMapping”
class=”org.springframework.web.servlet.handler.SimpleUrlHandlerMapping”>

        <property name=”mappings”>
<props>
<prop key=”/plafondService.httpInvoker”>
httpExporterPlafondService
</prop>
. . .
</props>
</property>
</bean>

</beans>
```

Figura 6.19 - Definição dos endpoints

A abordagem adoptada revelou-se bastante eficaz no processo de construção dos serviços, pois verificou-se que a utilização da metodologia para a definição de *Web Services* é efectiva no que respeita a:

- Facilidade de realização de testes de integração com ferramentas existentes para testar w*eb services* (SOAPUI[35]);
- Facilidade de versionamento dos serviços expostos.

6.2.5. Programação orientada a objectos

Uma característica importante destes serviços é a utilização de AOP (*Aspect-oriented programming*). A AOP fornece a capacidade de separar partes do código mediante a sua importância para a aplicação. Quando se está a configurar uma determinada versão existe um conjunto de validações que têm de ser realizadas. Para se perceber melhor este funcionamento é apresentado de seguida um exemplo prático. Neste exemplo o objectivo é validar se uma determinada versão, quando se está a configurar, está num estado que seja permitido realizar alterações nessa versão. Na figura 6.20 são definidas as configurações dos vários aspectos.

[35] SoapUI é uma ferramenta *open source* escrita em Java cuja principal função é consumir e testar web services [62].

```
<!−−=== AOP DEFINITIONS: INTERCEPTOR AND ASPECT CONFIG===−−>
<aop:aspectj−autoproxy/>
. . .
<bean id=”lcvinterceptor”class=”pt.ptinovacao.cm.aop.LCVInterceptor”>
<property name=”lcvServiceController”ref=”lcvServiceController”/>
</bean>

. . .
```

Figura 6.20 - Configuração AOP

Após configurados os vários aspectos é demonstrado um exemplo de uma implementação. Na figura 6.21 é definido um aspecto com um *pointcut*. Neste caso, o *pointcut* define que antes da execução de qualquer operação do serviço (*pt.ptinovacao.cm.service.impl.*.*) que tenha como argumento um *request*, será executada uma validação, neste caso, verificar que uma versão está no estado que permite a escrita. Se for permitido a implementação do serviço, este é executado, caso contrário é lançada uma excepção indicando o erro ao utilizador.

```
@Aspect
public class LCVInterceptor implements Ordered {
private static final int ASPECT ORDER=1;
ILCVServicePolicyControllerlcvServiceController;
public void setLcvServiceController(ILCVServicePolicyControllerlcvServiceController){
this.lcvServiceController=lcvServiceController;
}
public int getOrder(){
            return ASPECT ORDER;
}
@Before(”execution(_pt.ptinovacao.cm.service.impl._._
(pt.ptinovacao.cm.entities.Request+))”+”and args (request)”)
public void before(Request request) throws Throwable{
String key= request.getVersion().getVersion()
+”−−”+request.getVersion().ge tScope();
//Verificar se a versao pode ou nao ser escrita
if (request instanceof ModifyConfigurationRequest
&&!lcvServiceController.version IsWritable (request.getVersion(),request.getUse r())){
throw new LCVException(”version cannot be written!”);
}
String versionUUID=lcvServiceController.ge tVersionUuid (request.ge tVersion());
request.getVe rsion().setUuid (versionUUID);

}
```

Figura 6.21 - Implementação de um aspecto

Esta abordagem permitiu ter uma melhor estruturação de todo o código, separando-se aspectos como validações em métodos próprios, sendo interceptado cada pedido a estes serviços.

6.2.6. Integração dos Serviços GWT e Serviços Base

Depois da definição dos serviços base e de os expor através de um protocolo proprietário do *Spring* (*Spring's Remote*) o cliente (Serviços-GWT) tem que conhecer estes serviços. Para isto acontecer algumas configurações têm que ser feitas.

Passo 1 - Na inicialização da *servlet* dos Serviços-GWT no ficheiro *web.xml* deverá ser carregado o contexto *Spring* (figura 6.22).

```
. . .
<context−param>
  <param−name>contextConfigLocation</param−name>
  <param−value>classpath:gwt−rpc−services.xml</param−value>
</context−param>

. . .
```

Figura 6.22 - Spring's context

Passo 2 - Para definir o serviço remoto no *Spring* define-se no ficheiro gwt-rpc-services.xml o caminho para o Serviço Base (figura 6.23).

```
. . .
<!−−================== Proxy s e r v i c e s ====================−−>
<bean id="httpInvokerLCVService"
class="org.springframework.remot ing.httpinvoker.HttpInvokerProxyFactoryBean">
<property name="serviceUrl">
<value>
${cm.services.addr}/${lcvService.httpInvoker.ctx}
</value>
</property>
<property name="serviceInterface">
<value>
pt.ptinovacao.cm.service.ILCVService
</value>
</property>
</bean>

. . .
```

Figura 6.23 - Spring's context

Passo 3 - Por fim, no ficheiro *remoteserver.properties* define-se o caminho para os serviços (figura 6.24).

```
#Endereco do servidor onde se encontram os Servicos Base
cm.services.addr=http://localhost:8080/cm−services
#Definicao dos endpoints

lcvService.httpInvoker.ctx=lcvService.httpInvoker
```

Figura 6.24 - *Spring's context*

A partir deste momento os Serviços-GWT conseguem invocar os serviços base. A única tarefa dos Serviços-GWT é a de mapear os modelos do Cliente GWT para os DTO's gerados anteriormente, que são partilhados pelos Serviços-GWT e pelos Serviços base, e invocar o serviço base correspondente à acção do utilizador. Na resposta o processo é o inverso, mapeando os dados da resposta da invocação ao serviço base para o modelo do Cliente GWT, que será apresentado ao utilizador.

6.2.7. Lógica de Controlo de Versões

A arquitectura definida para integração entre os sistemas faz com que surja um modelo passivo unificado (CDM – *Common Data Model*), sendo que a aplicação de configuração passa a ser abstraída da existência de dois domínios disjuntos (NGIN Care e NGIN Core), e as configurações são feitas sobre o CDM. Numa arquitectura *"CORE only"*, a LCV (Lógica de Controlo de Versões), definia duas fases na

gestão de uma versão de configuração: Transferência e Activação. Para uma arquitectura *"NGIN Core + NGIN Care"*, passa-se a definir uma fase adicional onde é feita a transformação dos dados do CDM para o modelo passivo NGIN Care e para o modelo passivo NGIN Core. O mecanismo encarregue desta transformação é denominado de *"Config Splitter"*, e é um módulo interno à LCV. As operações *"high-level"* da LCV abstraem o utilizador que gere o ciclo de vida de uma versão da existência deste passo intermédio de transformação. A referida transformação dos dados é uma fase do ciclo de vida de uma versão que precede uma transferência.

6.2.7.1. Arquitectura da LCV

A LCV é um módulo orquestrador da solução NGIN que guarda a informação de topologia das plataformas constituintes de uma determinada implementação. A figura 6.25 detalha a arquitectura da LCV.

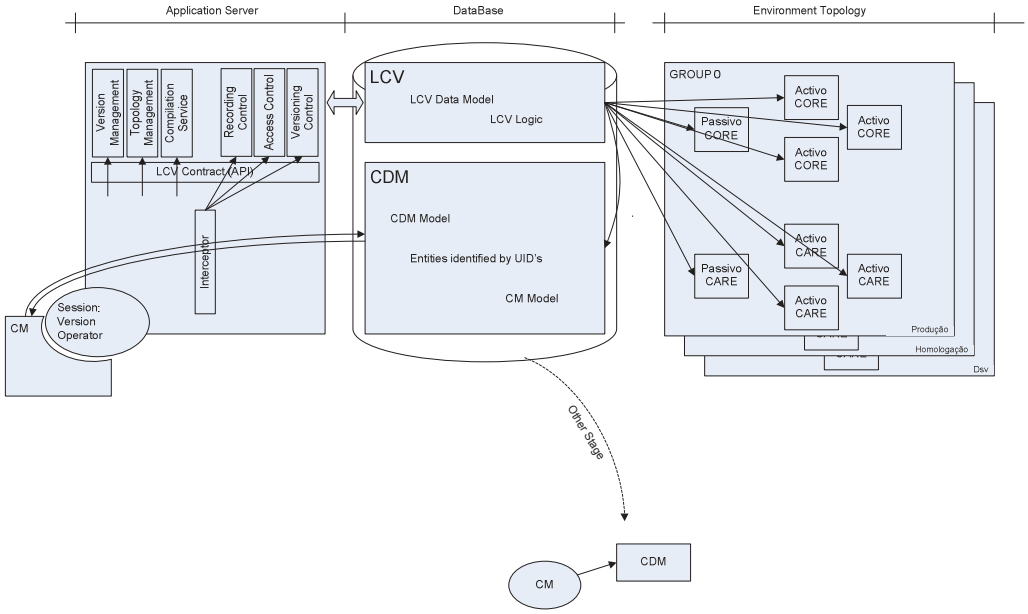

Figura 6.25 - Detalhes da arquitectura da LCV

Esta plataforma de controlo de versões, permite ao utilizador definir o comportamento desejado para o mecanismo de versionamento de configurações, e também das respectivas transferências e activações em ambientes de produção.

A LCV tem um papel de "*middleware*", entre a edição de uma versão de configuração e a sua real utilização em ambientes finais de utilização, disponibilizando uma API de interacção com as operações que o GUI do CM fornece e que se encaixam neste contexto de gestão de versões. Assim, esta camada permite um acesso normalizado e remoto aos ambientes passivos, dados esses já transformados pela ferramenta de ETL (*splitter*), e executar as suas funções mais importantes, tais como:

- Criar/copiar novas versões base de configuração e apagá-las local e remotamente;
- Transferir e activar versões de configuração dos ambientes passivos para os activos;

- Consulta de informação contextual, como por exemplo, saber qual a versão que está activa nos ambientes activos ou qual a versão que estava activa em determinada altura, entre outras.

É importante referir que a LCV é assente sobre a base de dados Oracle e toda a sua lógica é definida em PL/SQL. É também na LCV que podem ser configuradas as entidades (módulos) que agrupam as tabelas consoante o seu contexto da área de negócio, filtrados por uma versão e operador (âmbito de serviço). São essencialmente estes parâmetros que permitem diferenciar configurações diferentes por cliente, consoante os seus propósitos e negócios.

A LCV fornece um conjunto de funcionalidades transversais associadas ao versionamento das configurações, que funciona independentemente da topologia configurada.

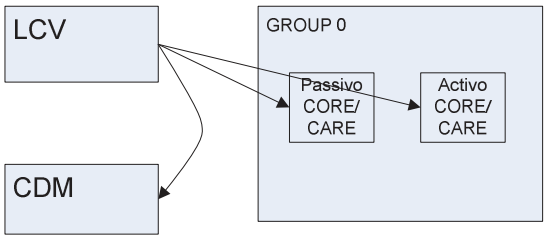

Figura 6.26 - Topologia mínima

As plataformas constituintes podem ser de diversos tipos, sendo que para cada tipo diferente de plataforma, define-se o respectivo *"activation handler"*.

A fase de transformação extrai os dados do CDM e, segundo um conjunto de regras configuráveis que específica uma transformada a aplicar aos dados, encarrega-se da criação dos dados (transformados) nas plataformas destino. As plataformas que podem ser destinos de transformação são: PASSIVO CARE e PASSIVO CORE.

Após fase de transformação, é então despoletada a fase de transferência – que ocorre dos destinos de transformação para os destinos de transferência (dos modelos passivos para os modelos activos).

6.2.7.2. Funcionalidades da LCV

Partindo do pressuposto que a LCV está ciente de todas as operações do CM, é possível fornecer um conjunto de funcionalidades transversais a todo o sistema. Na parte de gestão de versões é possível, criar, copiar, bloquear/desbloquear, fechar, remover, sincronizar, activar, salvar, exportar, carregar, importar, agendar transferências, agendar activações de versões. A LCV permite também uma gestão de topologias, serviços de compilação, mecanismos de *recording*, controlo de acessos e controlo de versões.

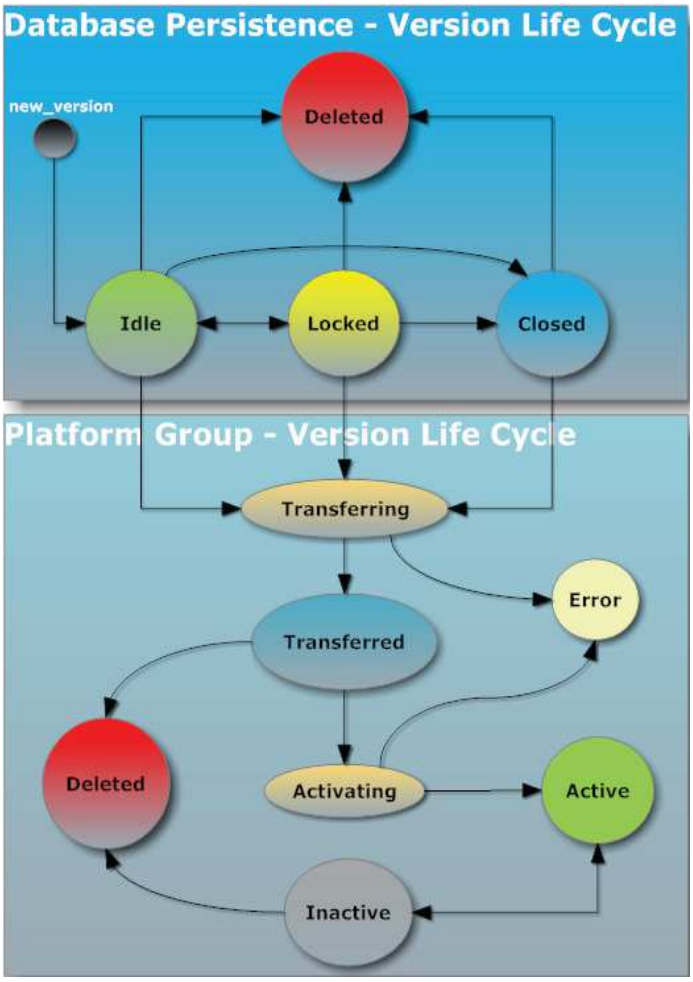

Figura 6.27 - Ciclo de Vida de uma Versão

Analisando a figura 6.27 acima apresentada, são descritos os estados pelos quais um ciclo de vida pode passar. As versões têm dois ciclos de vida, um no contexto de base de dados e outro no contexto dos grupos de plataformas. No contexto de BD uma versão pode passar por os seguintes estados:

Estado *Idle* – Estado para uma versão em desenvolvimento. Uma versão que esteja no estado *idle* pode ser editada concorrentemente por diversos utilizadores. Quando é efectuada uma cópia de uma versão ou quando é criada uma nova versão, ela é criada no estado *idle*.

Estado *Locked* – O comportamento do estado é parametrizável através da alteração de uma propriedade nas definições que a LCV disponibiliza. Existem dois comportamentos distintos para o estado *locked*. Num caso o utilizador que bloqueou a versão continua a poder alterá-la (exclusivamente), no outro caso as versões no estado *locked* não podem ser alteradas mesmo por quem as bloqueou.

Estado *Closed* – O estado *closed* é um estado que impossibilita futuras alterações a uma determinada versão. Quando uma versão chega a este estado já não pode ser modificada. Pode apenas ser apagada.

Estado *Deleting* – Estado transitório pelo qual uma versão passa enquanto está a ser eliminada.

Estado *Deleted* – Uma versão *deleted* é uma versão que deixou de existir, foi portanto apagada. Sempre que se executa uma operação de *delete*, é possível indicar que se pretende fazer "*purge*" à versão. Um *delete* que ocorra no modo "*purge*" não deixa o registo de histórico que prova que a versão existiu. Além de eliminar a versão, elimina ainda o registo de controlo dessa versão.

Como foi dito cada versão pode ter um ciclo de vida diferenciado para cada grupo que é destino de uma transferência. A parte do ciclo de vida que é independente do grupo de plataformas é denominada de *database persistence – version life cycle*. A parte do ciclo de vida que depende do grupo de plataformas é denominada de *platform group – version life cycle*. Os diferentes estados da versão numa plataforma são descritos a seguir:

Estado *Transforming* – Estado transitório pelo qual uma versão passa enquanto o processo de ETL dos dados está a decorrer (CDM →PASSIVO CORE e CARE).

Estado *Transfering* – Depois de uma versão transformada, é iniciada a transferência de uma versão e colocada no estado *transfering*.

Estado *Error* – Qualquer erro que ocorra num dos estados *transforming, transfering, activating* e *deleting* faz com que a versão seja marcada com um estado de *error*.

Estado *Synchronized* – A versão fica neste estado depois de ser transformada e transferida com sucesso. É a partir deste estado que é possível activar uma versão.

Estado *Activating* – Estado intermédio pelo qual uma versão passa durante o período de activação. Este período de activação depende dos valores configurados nas propriedades relativas ao mecanismo de activação.

Estado *Active* – Só pode existir simultaneamente uma versão no estado *active* por cada grupo de plataformas (por operador). Uma versão no estado *active* indica que é sobre as configurações dessa versão que estão a ser processadas as sessões/operações que entram na plataforma NGIN.

Estado *Inactive* – Sempre que uma versão é colocada no estado *active*, a que estava anteriormente neste estado é colocada no estado *inactive*. O número de versões que ficam no estado *inactive* (em histórico) é configurável a partir de uma propriedade de parametrização da LCV. Versões que estão no estado *inactive* podem ser reactivadas sem que haja a necessidade de passar por novo processo de transformação e transferência (*Synchronize*).

6.2.8. Config Splitting

Arquitecturalmente, o "config splitter" é um módulo da LCV que disponibiliza um método denominado *lcv_transform_version.* O *config splitter* aprovisiona um conjunto de regras de transformação que têm que ser independentes dos seguintes detalhes:

- Modelo canónico (CDM) de origem;
- Complexidade dos dados persistidos no CDM origem;

- Devem contemplar 2 tipos de plataformas destino para os dados transformados: PASSIVO CORE e PASSIVO CARE;
- Devem filtrar os dados a transformar tendo em conta o contexto da transformação;
- As regras de transformação são definidas módulo a módulo;
- Existe uma *pool* de métodos que deve ser utilizada para facilitar o processo de transformação (e *splitting*) dos dados, o *Helper Methods Pool*.

A figura 6.28 mostra a arquitectura do *config splitter* na LCV.

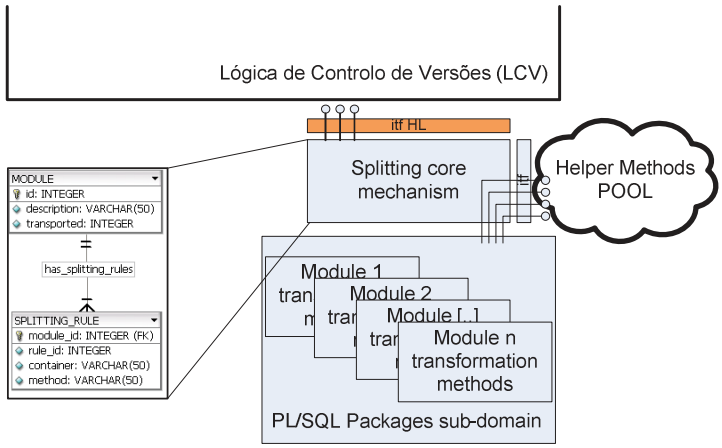

Figura 6.28 - Módulo *Splitter* da LCV

É possível arranjar uma sintaxe tão complexa quanto se queira para se definir um processo de ETL. Atendendo a que a linguagem PL/SQL, é a linguagem mais familiar ao público-alvo que definirá as regras de transformação, é sobre esta sintaxe que as regras de transformação são definidas. A escrita das regras não deve ser feita de livre arbítrio. Deverá respeitar os pressupostos apresentados anteriormente e seguir o conjunto de regras apresentadas ao longo desta secção. Partindo de um contexto de transformação e da configuração da topologia, as regras de transformação devem ser capazes de extrair os dados necessários e suficientes aos modelos CARE+CORE que permitem que a lógica de negócio se comporte tal como é especificado na ferramenta de configuração. A figura 6.29 apresenta o processo de *splitting* desde os *inputs* até aos *outputs* inseridos nos modelos passivos.

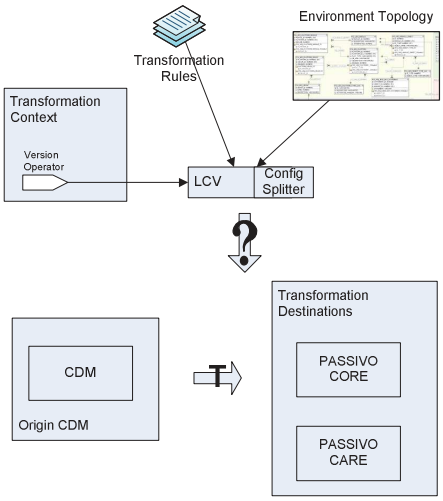

Figura 6.29 - Inputs/Outputs das *RULES* de transformação

6.3.Interface com o utilizador

O CM encontra-se integrado com o SCA (Sistema de controlo de acessos) e o utilizador para poder aceder à aplicação terá de se autenticar com sucesso usando a página apresentada na figura 6.30.

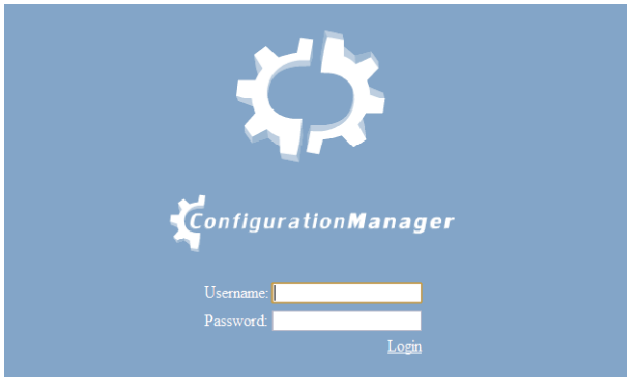

Figura 6.30 - Ecrã de autenticação

Este componente externo ao CM será utilizado para efeitos de autenticação e autorização. Com base em consultas efectuadas ao SCA será determinada a validade das credenciais fornecidas pelo utilizador bem como informação relativa às funcionalidades que tem acesso na aplicação CM. O seu mecanismo de autenticação permite restringir o acesso do utilizador a determinadas áreas de configuração bem como quais as permissões de edição que pode ter. Depois de feita a autenticação com sucesso, o utilizador é redireccionado para a página de gestão de versões. No cabeçalho da aplicação ficará a informação de qual o utilizador que se autenticou através desta página. Ainda relativamente a autenticação, o utilizador depois de entrar na aplicação terá um tempo de inactividade controlado, a partir do qual a sua sessão expirará sendo aí necessário que se volte a autenticar na aplicação.

6.3.1. Áreas de trabalho

A interface gráfica do CM encontra-se dividida em duas partes distintas. O cabeçalho, onde se encontram os botões de acesso às áreas de trabalho, o botão de saída da aplicação e indicação do

utilizador autenticado na aplicação, a área de trabalho propriamente dita, que varia consoante a escolha entre a Gestão de versões e a Gestão de topologia. São estas as duas grandes áreas de trabalho que o CM disponibiliza, sendo que a Gestão de versões encontra-se dividida em mais duas áreas fundamentais no processo de configuração, Gestão de configurações e Gestão de transferências. Seguindo uma ordem lógica do processo normal de configuração, a primeira área do CM é a de Gestão de Topologia seguindo-se a de Gestão de versões, dentro da qual se encontram as áreas de Gestão de configurações e Gestão de transferências. Para dar uma maior flexibilidade ao utilizador no uso destas áreas de gestão, o CM apresenta-as numa estrutura tabular à medida que o utilizador as for abrindo, permitindo-lhe manter as configurações abertas.

Na situação que existe uma mudança no contexto de configuração não faz sentido que sejam mantidas abertas janelas de configuração de outro contexto, por isso, todas serão fechadas mantendo-se a área de Gestão de versões e Gestão de topologia, caso estas estejam abertas. O utilizador será notificado através de uma mensagem do fecho das páginas de configuração do contexto anterior.

A área de Gestão da topologia, apresentada na Figura 6.31, permite a configuração e visualização das plataformas e grupos de plataformas para onde podem ser transportadas, transferidas e activadas as configurações. Aqui podem ser configuradas quais as plataformas para onde se pretende transportar e activar as versões de configuração, bem como agrupa-las de acordo com a arquitectura da solução que se está a configurar.

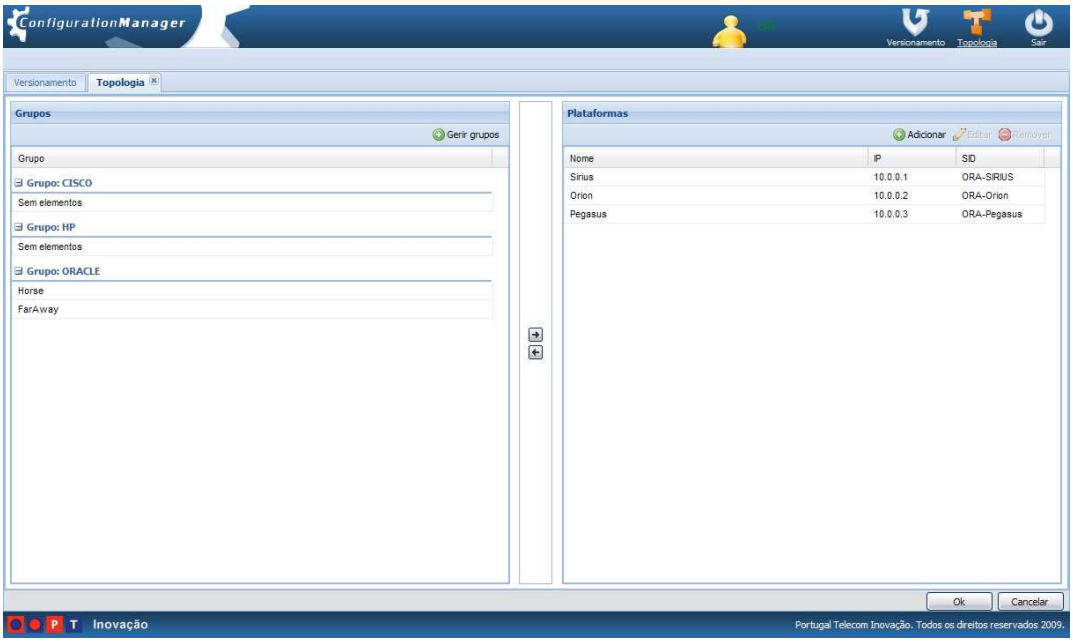

Figura 6.31 – Gestão de topologia

A área de Gestão de versões, onde são geridas as várias versões de configuração desde a sua criação até ao seu transporte, transferência e activação para plataformas externas, pode ser vista na Figura 6.32 e apresenta as versões disponíveis organizadas segundo um âmbito definido pelo utilizador, neste caso o âmbito apresentado é o operador de telecomunicações.

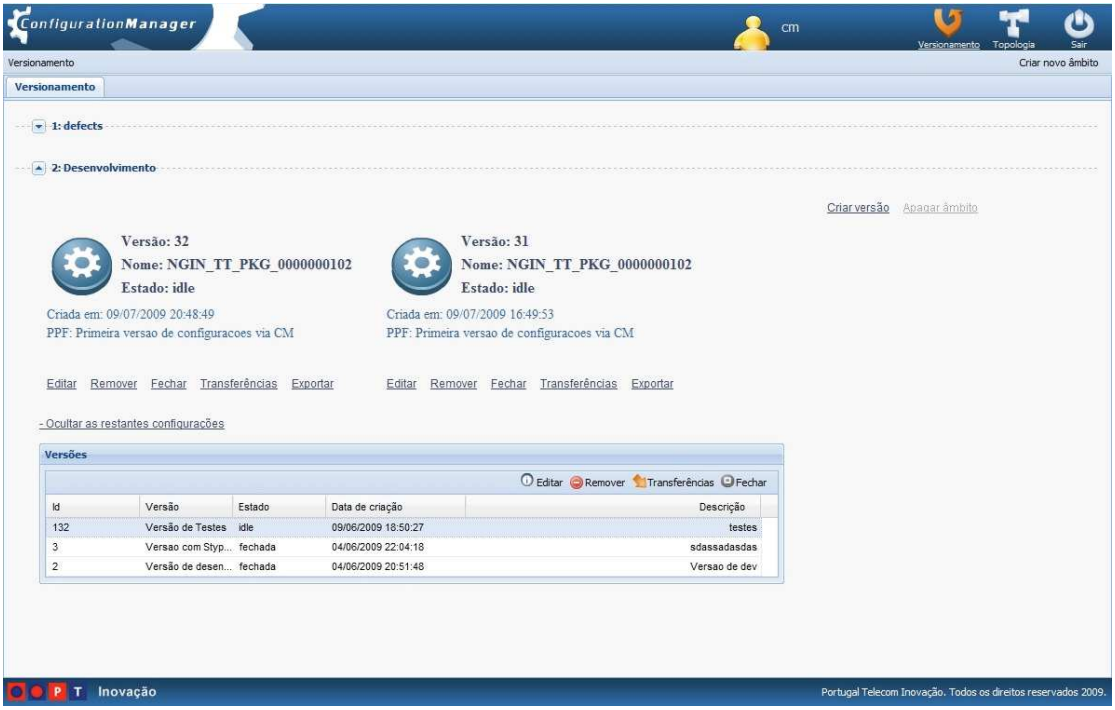

Figura 6.32 – Gestão de Versões

Uma vez identificado o âmbito e a versão de configuração, é possível prosseguir para a sua área de Gestão de configurações, como apresentado na Figura 6.33, ou para a sua área de Gestão das transferências, como apresentado na Figura 6.34.

Na área de gestão de configurações, o utilizador poderá configurar os diferentes tipos de entidades disponíveis para o seu âmbito de configuração podendo criar novas configurações ou explorar as já existentes. Ao explorar uma determinada entidade, serão apresentadas as entidades existentes no sistema, podendo estas ser editadas ou removidas.

Graças à tabulação, podem ser abertas várias configurações de diferentes entidades em simultâneo.

O ponto central de configuração é a entidade perfis. O perfil ou produto ou oferta comercial, é a entidade principal que tem na sua composição um conjunto de outras entidades, que na sua maioria isoladamente e independentes do perfil acabam por não ter significado. No entanto são essenciais para criar e distinguir as várias ofertas comerciais que cada operadora poderá ter. A este conjunto de ofertas comerciais podemos definir como um catálogo de produtos e serviços.

Figura 6.33 – Gestão de Configurações

Após efectuar as alterações na configuração de uma versão, ou indo directamente para a sua área de Gestão de transferências, figura 6.34, é possível visualizar as transferências dessa versão que já foram feitas para os diversos grupos de plataformas sendo possível aí gerir essas transferências. Esta área depende das configurações efectuadas ao nível da topologia, na primeira área apresentada.

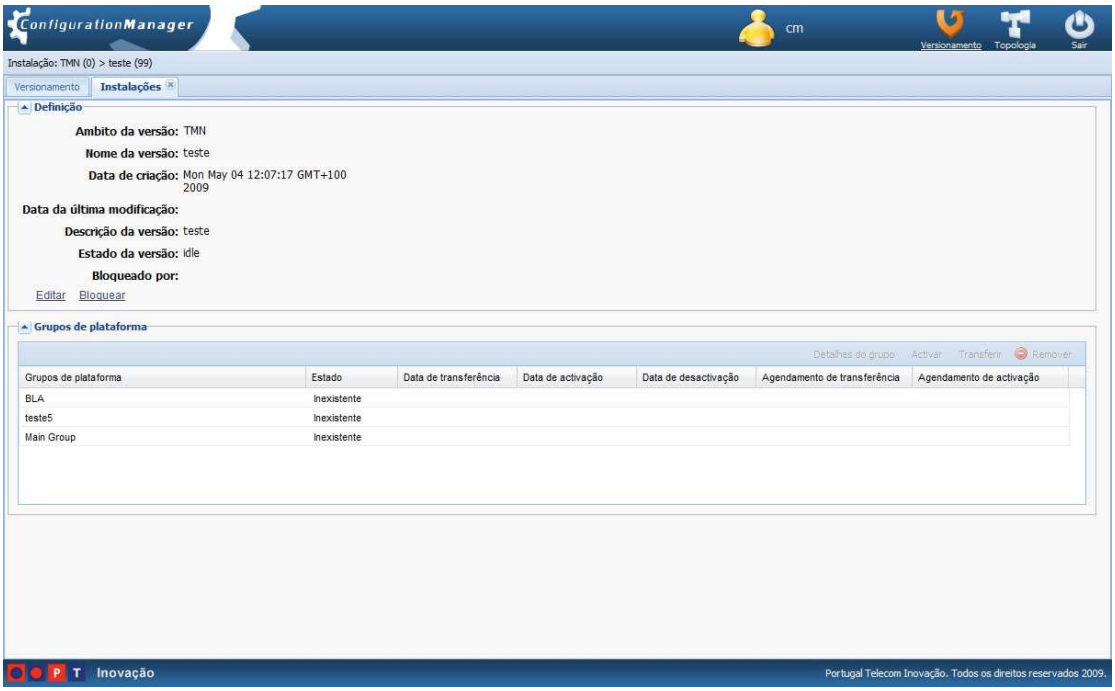

Figura 6.34 – Gestão de Transferências

6.3.2. *Wizards/Steps*

Na área de Gestão de configurações, o CM dispõe de um mecanismo de auxílio ao processo de configuração que consiste em *wizards* onde o utilizador é conduzido por um conjunto de formulários interligados e que o guiam através da configuração de determinado aspecto de negócio do CM.

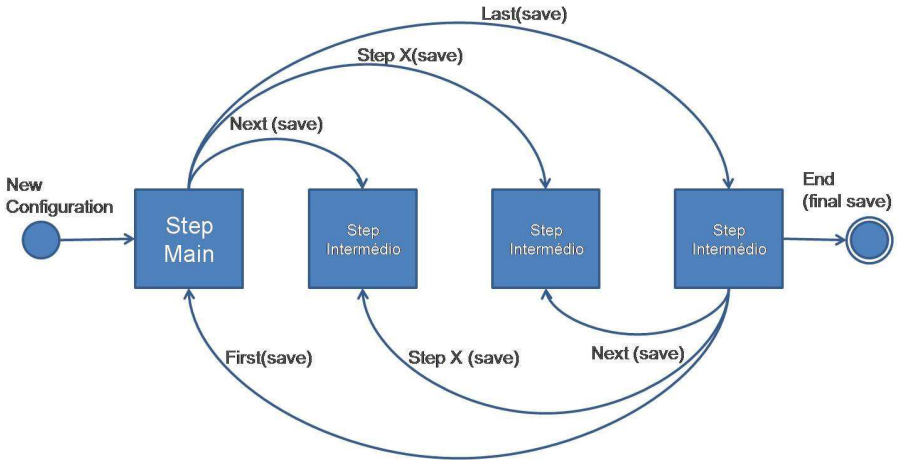

Figura 6.35 - Configuração passo-a-passo

Este processo passo-a-passo é composto por um passo inicial onde é iniciada a configuração, sendo depois seguido por vários passos intermédios até à conclusão da configuração. É possível ao configurador navegar entre os passos pelos quais já passou, uma vez que as configurações vão sendo persistidas pelo CM (auto-save). Além da persistência automática, é possível gravar a configuração havendo para isso a operação disponível nos formulários, bem como botões de navegação para ir avançando ou recuando nos passos da configuração.

6.3.3. Validação Formulários

Graças à tecnologia usada no desenvolvimento da aplicação, todos os formulários dispõem de mecanismos de validação dos dados introduzidos (formato, tamanho máximo permitido e obrigatoriedade) sendo o utilizador notificado através de *tooltips* (mensagens disponibilizadas ao passar o rato sobre os campos do formulário) e com mensagens de erros que existam quando a configuração é guardada. Este comportamento é transversal a toda a aplicação.

6.3.4. Operações sobre a configuração

Na aplicação, as operações sobre os dados de configuração são comuns às diversas entidades e encontram-se bem identificadas no formulário a que dizem respeito. Se determinada operação for dirigida a um registo em particular, o ícone correspondente aparecerá desactivo sendo necessário seleccionar primeiro o registo que se pretende alterar para que o botão passe a estar activo. No caso de a operação não ser referente a nenhum registo em particular, o ícone correspondente surgirá activo sem que seja necessária a selecção de registos em particular. Tipicamente, as operações incluem criação, edição e remoção de configurações.

A operação de Criação é identificada pelo ícone sendo este também usado na operação de Associação. No primeiro caso será aberto o formulário que permitirá a criação da entidade de configuração, enquanto no segundo caso, será aberta uma janela com a listagem dos itens a associar à configuração inicial.

A operação de Edição é identificada pelo ícone e o formulário que será apresentado ao utilizador será o mesmo que o da operação de criação. Para efectuar esta operação, será necessário previamente seleccionar o registo que se prete nde alterar.

A operação de Remoção é identificada pelo ícone e permite ao utilizador remover o registo que previamente seleccionou. Este ícone é partilhado pela operação de Desassociação, que não irá remover o registo da mesma forma que a operação de Remoção, mas sim eliminar a associação que exista entre as entidades. Será pedida confirmação ao utilizador da realização destas operações através de uma mensagem de acordo com o contexto onde está a ser feita a remoção.

A operação de ”Criar como” é identificada pelo ícone e permite ao utilizador criar uma entidade com base noutra, ficando a nova entidade inicialmente, com as mesmas definições da original. Este mecanismo é bastante útil quando existe a necessidade de criar entidades semelhantes a outras já configuradas.

A operação de Exportação é identificada pelo ícone e permite ao utilizador exportar uma entidade ou um conjunto de entidades, bem como as entidades dependentes desta(s).

Para finalizar as configurações, o utilizador poderá usar o botão “Concluir”, podendo no entanto apenas gravar a configuração através do botão “Salvar”. Caso pretenda reverter as alterações efectuadas deverá usar o botão “Cancelar” do respectivo *Wizard*.

6.3.5. Navegação horizontal/vertical

A navegação horizontal, corresponde à sequência de configurações dentro do contexto de uma mesma entidade, enquanto que a navegação vertical, corresponde à configuração de um outra entidade, partindo do contexto de uma outra entidade. A figura 6.36 apresenta um exemplo dos dois tipos de navegação.

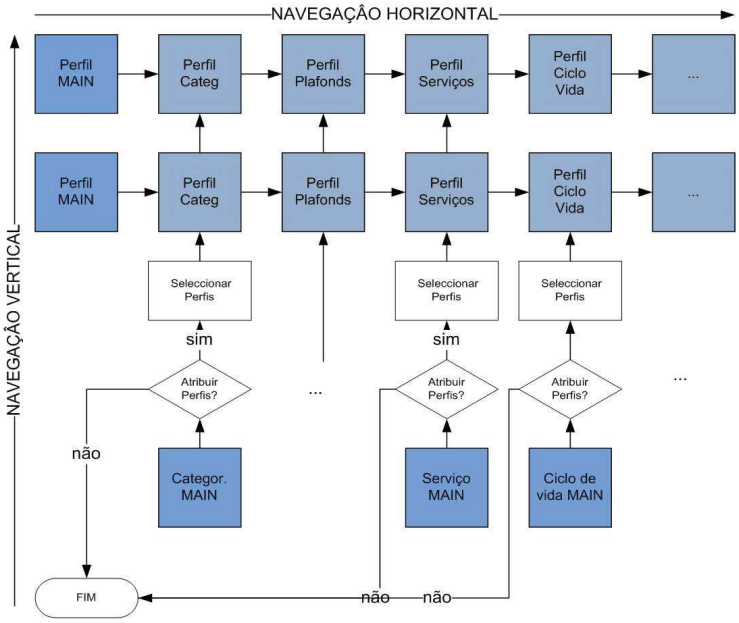

Figura 6.36 - Navegação horizontal e vertical

A sequência de configuração dos *wizards*, ou seja, passo-a-passo, corresponde a uma navegação horizontal, uma vez que vão sendo percorridos vários aspectos da configuração sempre sobre a mesma entidade. De cada vez que essa configuração se cruza com outra entidade e dentro do *wizard* se fazem configurações da mesma, sem sair do contexto horizontal, estamos perante uma navegação vertical sobre a configuração. Um exemplo para melhor compreender esta questão poderá ser a configuração de perfil, composta por vários passos de configuração (navegação horizontal), sendo que por exemplo no passo de configuração de categorias é possível, além da associação, fazer gestão dessas categorias (navegação vertical).

6.3.6. Navigator

Com a configuração das entidades de negócio é criado um mapa de interdependência entre as diversas entidades, sendo esse mapa por vezes tão complexo quanto o número de entidades que se interligam entre si. Para facilitar a consulta de informação, o CM dispõe de um mecanismo de navegação pelas configurações que obtém as listas das entidades de configuração e respectivas relações de interdependência.

A possibilidade de explorar as relações entre as diversas entidades é fundamental na medida em que facilita o processo de análise de dependências e impactos sempre que se altera a definição de uma instância de um determinado tipo de entidade.

O *navigator* tem por base o conceito de apresentação da informação organizada em forma de árvore permitindo ainda que na exploração da árvore de dependências sejam aplicados filtros que restringem as instâncias apresentadas com base em critérios definidos. O *navigator* dispõe ainda de funcionalidades integradas de edição e remoção de instâncias. A figura 6.37 apresenta um exemplo do *navigator*, neste caso é possível ver as interdependências entre entidades a partir de um perfil.

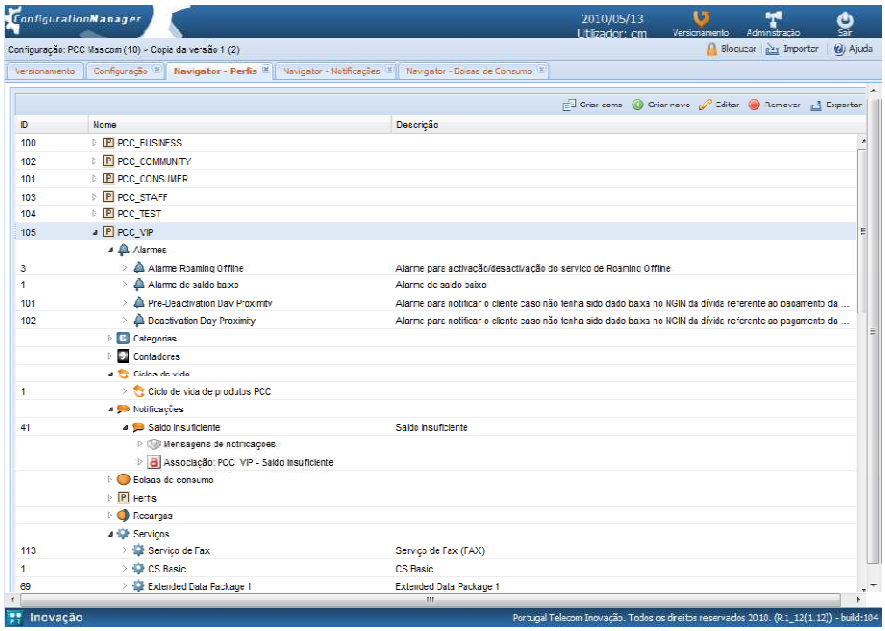

Figura 6.37 - Árvore de navegação da entidade perfis

A figura 6.37 permite também observar os perfis/ofertas comerciais configurados para esta versão sendo que no caso da oferta PCC_VIP é possível verificar que esta tem um conjunto de alarmes associados, contadores, notificações, bolsas de consumo, recargas, serviços. A entidade perfis aqui apresentada serve essencialmente para configurar as características que um cliente quando mudar de oferta para a oferta comercial PCC_VIP ou então mudar dessa oferta comercial para outra configurada vai ter. Também a partir da figura 6.37 é possível verificar um conjunto de ofertas comerciais definidas para esta versão de configuração.

6.3.7. *Edição massiva*

O CM dispõe de outro mecanismo de agilização que neste caso possibilita a edição em massa de configurações. Recorrendo a um marcador que pode ser aplicado a uma configuração ou conjunto de configurações, é possível fazer reutilização dessas configurações em várias outras instâncias. Estes marcadores podem ser criados e aplicados a passos de configuração de um *Wizard* e a sub-grupos de configurações nesses passos. A partir da configuração das propriedades de associação num passo poderá ser criado um marcador para a mesma, de modo a reutilizá-la noutra instância da mesma entidade. A visualização dos marcadores existentes e quais as entidades que este associa é feita através de um navegador de marcadores. Neste *navigator* é possível a edição dos marcadores existentes, afectando assim todas as entidades que tenham sido marcadas proporcionando uma alteração em massa de configurações. A figura 6.38 é um exemplo desta funcionalidade.

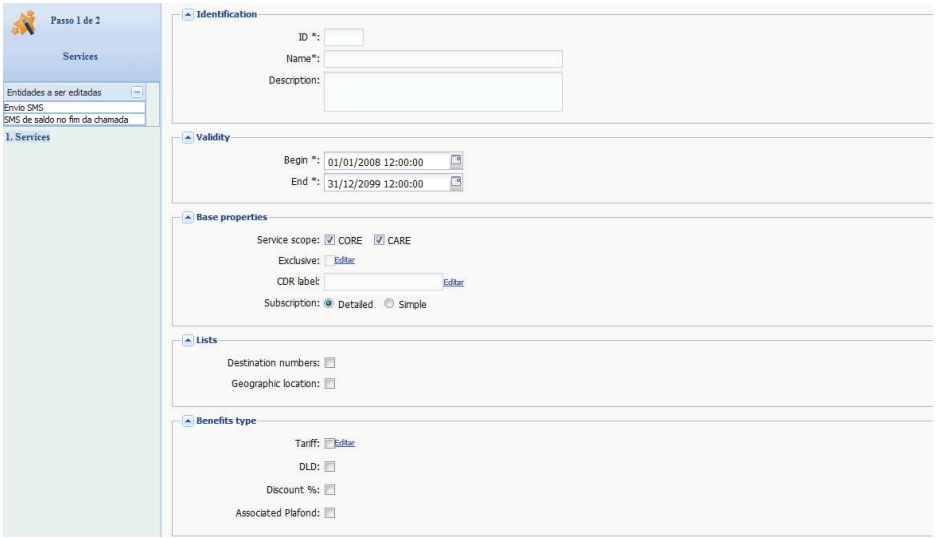

Figura 6.38 – Edição em massa

6.3.8. *Mecanismo de importação e exportação*

A importação e a exportação são mecanismos essenciais para a utilização do CM. Quando é feita uma configuração de uma entidade qualquer na aplicação pode haver a necessidade de exportar essa configuração para outra versão de configuração, ou até para outro cliente. Se existe a funcionalidade de exportar uma entidade tem de existir a funcionalidade de a importar. De salientar que este mecanismo de exportação pode ser feito sobre uma entidade em particular ou sobre uma versão total de configuração.

No anexo 2 é apresentado um pequeno cenário de configuração onde é demonstrado o uso da ferramenta *Configuration Manager* num cenário concreto.

.

7. Conclusões e Trabalho futuro

7.1. Conclusões

O objectivo deste trabalho é contribuir de uma forma prática para o enriquecimento da solução baseada em redes inteligentes da Portugal Telecom. No mundo empresarial, e em especial nas empresas de telecomunicações, a competição é elevada e é necessário a todo o instante obter novas formas de tornar as soluções disponibilizadas cada vez mais completas, manipuláveis e à medida dos requisitos das operadoras. Esse trabalho é bastante difícil. Desenvolver *software* funcional que responda às necessidades que são propostas quase em tempo real é um grande desafio.

O desenvolvimento de uma aplicação de validação e automatização de testes é cada vez mais necessária numa empresa e muitas vezes, como ficou demonstrado com o estudo do estado da arte, deve ser usada em várias fases do desenvolvimento do *software* e não só no final desse processo de desenvolvimento. Se existir uma interligação entre o desenvolvimento e a execução dos testes é mais provável que sejam detectados e resolvidos os erros encontrados, do que se estes só forem encontrados no final de todo o processo de desenvolvimento. O estado da arte aqui apresentado nesta área de testes foi bastante útil pois enquadra o leitor nesta temática, servindo também, e em especial, para a percepção das hipóteses de desenvolvimento e extensão da ferramenta *TestBeds*.

A ferramenta *TestBeds* devido a sua capacidade de validação de *software* e avaliação de retrocompatibilidade de versões suprimiu uma das grandes lacunas existentes na PT Inovação, pois uma das principais queixas dos clientes e das equipas de testes externas ao departamento, era que aquando do lançamento de um novo produto de *software,* áreas que funcionavam de uma forma depois dessa alteração passavam a ter um comportamento diferente. Com a capacidade que a ferramenta possui de executar testes de regressão, esses problemas foram reduzidos em muito. A ferramenta tornou a solução NGIN, mais capaz, fiável, disponibilizada com menos erros, e com a capacidade de validar se os novos desenvolvimentos têm quebra de comportamentos.

A ferramenta desenvolvida deixou de fora muitas áreas descritas no estado da arte, no entanto, o âmbito desejado para a ferramenta não tinha como objectivo a preocupação com essas áreas. Questões de testes de carga, por exemplo, não foram equacionadas pois para isso já existiam ferramentas.

Como conclusão em relação à ferramenta *TestBeds,* o seu desenvolvimento foi muito importante tanto para mim como para a empresa, no entanto a sua evolução é quase infindável, sendo assim uma excelente matéria-prima para o desenvolvimento de novos projectos, de forma a fortalecer e capacitar a ferramenta cada vez mais.

Como foi explicado ao longo deste documento, este trabalho de mestrado é composto pelo estudo do estado da arte e pelo desenvolvimento de duas ferramentas com objectivos distintos, que têm como ponto de ligação a solução NGIN.

O desenvolvimento da aplicação de configuração teve como grande objectivo permitir a configuração das entidades e regras de negócio que vão determinar o comportamento do sistema NGIN,

formando assim um conjunto de ofertas comerciais diferenciadas. Este conjunto de ofertas forma o catálogo de produtos e serviços de um determinado cliente.

Esta ferramenta surgiu de uma forma inevitável pois a evolução natural da solução NGIN deixou-a sem nenhuma ferramenta de configuração. E para que serve uma solução que depois não é possível de configurar de uma forma amigável? Configurar usando directamente o modelo de dados desde de 2004 que para a PT Inovação deixou de ser solução, por isso surge o projecto *Configuration Manager.* Este projecto devido à sua elevada dimensão foi aliciante para todos os intervenientes e culminou com um enorme sucesso na sua primeira utilização e instanciação numa operadora. Essa experiência aconteceu na Timor Telecom, e desde aí já outras operadoras se seguiram como a CVT (Operadora de telecomunicações Cabo-Verde), CST (Operadora de telecomunicações de São Tome e Príncipe), Meditel (Operadora de Marrocos), Mascom (operadora do Botswana), estando agora em fase de aceitação na TMN. O sucesso da ferramenta e a sua mais valia são evidentes, no entanto para os clientes exigentes são necessárias constantes evoluções, o que torna este projecto sempre apelativo e interessante para quem o desenvolve.

7.2. Trabalho futuro

Como trabalho futuro, a ferramenta *TestBeds*, deverá numa próxima versão, ter uma integração com o sistema de controlo de acessos para ser possível definir áreas de acesso diferentes mediante o tipo de utilizador que está a utilizar a ferramenta. A integração com a ferramenta de gestão de defeitos existente, sendo que essa integração deverá ser relativamente automática, ou seja, quando num teste for detectado um problema deverá de imediato ser elaborado um novo caso nesta ferramenta de gestão de defeitos. Outro aspecto a ter em conta é a forma como os resultados dos testes são apresentados, a imagem de uma ferramenta é cada vez mais importante e a questão dos relatórios não foi tida muito em conta nesta versão. Para além disto nesta área era importante existir mecanismos de estatísticas com geração de gráficos com percentagens de testes executados com sucesso, testes com problemas, que testes são mais vezes executados, etc. Um item também a ter em conta numa futura versão das *TestBeds,* é a melhoria nos mecanismos de selecção/inserção de informação de forma a minimizar erros de configuração. Alargar o âmbito da ferramenta para sub-componentes que não apenas a suite genérica, poderá também ser uma excelente área de expansão da ferramenta. Aliado a isto e tendo em conta o estado da arte apresentado existem vários tipos de testes que a ferramenta num futuro pode também abarcar.

Em relação à ferramenta, *Configuration Manager*, com o seu uso real muitos requisitos e melhorias foram apontadas, aliando-se a isso também a constante evolução da solução NGIN. Estes têm sido os dois grandes aspectos de evolução da ferramenta. O *Configuration Manager* tem de acompanhar as evoluções de negócio e tem de ir de encontro aos pedidos dos clientes, sendo apenas possível assim, o seu sucesso.

Materializando estes aspectos as principais evoluções com que o *Configuration Manager* se depara passam pelas constantes melhorias de performance, agilização de processos e requisitos de negócio motivados pela evolução natural da plataforma NGIN. Para além destes, a integração num portal de

ferramentas da Portugal Telecom, é um ponto para o futuro, pois com isto vai ficar mais facilmente acessível. A comparação entre versões de configuração é algo que o cliente tem insistido junto da equipa, pois existe a necessidade aquando da configuração de novas ofertas comerciais ou entidades, perceber quais foram as alterações que existiram aí, ou seja, é pretendido assim que a ferramenta tenha a capacidade de analisar e mostrar ao utilizador quais as diferenças entre duas configurações ou entidades configuradas. Um mecanismo que também tem sido exigido é capacitar a ferramenta com filtros e listagens complexas para que o utilizador consiga visualizar melhor apenas o que procura e não todo o espectro de configurações de uma determinada entidade. Sugestões e melhorias com menos expressão são também necessárias de acompanhar de forma a cada vez mais ir ao encontro do que o cliente pede e espera.

Referências bibliográficas

[1] – PT Inovação, [On-line] Disponível em http://www.ptinovacao.pt/ [acedido em Setembro 11, 2010]

[2] - C. Moura and V. Freitas *"I Conferência Nacional de Telecomunicações",* Aveiro, Portugal, [Abril 10-11, 1997], [On-line] Disponível em http://marco.uminho.pt/CCG/papers/serv-telecom-em-redes-intelig-abs.html [acedido em Agosto 12, 2010]

[3] - Sinalização por canal comum número 7 – Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/Sinaliza%C3%A7%C3%A3o_por_canal_comum_n%C3%BAmero_7 [acedido em Setembro 12, 2010]

[4] - Signaling transport - Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/SIGTRAN [acedido em Setembro 12, 2010]

[5] - Protocolo de iniciação de sessão - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/SIP [acedido em Setembro 12, 2010]

[6] – Um esboço sobre o processo de teste de software [On-line] Disponível em http://www.homologacao.php.ba.gov.br/prodeblog/?tag=testes-de-software [acedido em Setembro 18, 2010]

[7] - G. J. Myers, *The Art of Software Testing*. Wiley, New York, 1979.

[8] – I. Burnstein, *Practical Software Testing: A Process-Oriented Approach*. Springer Professional Computing, 2003.

[9] - XML - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/XML [acedido em Setembro 25, 2010]

[10] - Java - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o) [acedido em Setembro 25, 2010]

[11] – W3C - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/W3C [acedido em Setembro 25, 2010]

[12] - Google Web Toolkit – Plataforma Web da Google - Wiki P@P, [On-line] Disponível em http://wiki.portugal-a-programar.org/revistaprogramar_arquivo:19_edicao:intro-google-web-toolkit [acedido em Setembro 25, 2010]

[13] – Open Source - Wikipedia, [On-line] Disponível em http://portal-gestao.com/gestao/1414-open-source-open-innovation.html [acedido em Setembro 25, 2010]

[14] - JavaScript - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/JavaScript [acedido em Setembro 25, 2010]

[15] - HTML - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/HTML [acedido em Setembro 25, 2010]

[16] - CSS - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/Cascading_Style_Sheets [acedido em Setembro 25, 2010]

[17] - ETL - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/ETL [acedido em Outubro 2, 2010]

[18] - Castor - Wikipedia, [On-line] Disponível em http://www.castor.org [acedido em Outubro 2, 2010]

[19] - iBatis - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/IBATIS [acedido em Outubro 2, 2010]

[20] - R. S. Pressman, *Software Engineering - A Practitioner's Approach*. McGraw-Hill, 4 edition, 1997.

[21] - J. C. Maldonado, *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. PhD thesis, DCA/FEE/UNICAMP, Campinas, SP, July 1991.

[22] - *IEEE. Ieee standard glossary of software engineering terminology.* Standard 610.12, IEEE Press, 1990.

[23] – R.D. Craig, S. P. Jaskiel, *Systematic Software Testing*. Artech House Publishers, Boston, 2002.

[24] – M. Haug, E. W. Olsen, L. Consolini, *Software quality approaches: testing,verification, and validation: software best practice.* 1. Berlin: Springer, 2001.

[25] - D. E. Perry and G. E. Kaiser, *Adequate testing and object-oriented programming. Journal on Object-Oriented Programming,* pages 13–19, January/February 1990.

[26] - S. Fujiwara, Gf. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, *Test selection based on finite state models. IEEE Transactions on Software Engineering,* 17(6), June 1991.

[27] – Ciclo de vida do *software*, - [On-line] Disponível em http://engenhariadesoftware.blogspot.com/2007/02/ciclo-de-vida-do-software-parte-1.html [acedido em Outubro 9, 2010]

[28] - E. V. Berard. *Essays on Object-Oriented Software Engineering*, volume 1. Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1992.

[29] – J.F. Peters, W. Pedrycz, *Engenharia de software: teoria e prática.* Rio de Janeiro: Campus, 2001.

[30] - Walkthrough - Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/Walkthrough [acedido em Outubro 10, 2010]

[31] – A. Bartie, (2002). *Garantia de qualidade de software* (4ª ed.), Campus, São Paulo.

[32] - ABNT – Associação Brasileira de Normas Técnicas. (1994a). *NBR ISO 8402/1994: Gestão da qualidade e garantia da qualidade – Terminologia*, Rio de Janeiro, ABNT.

[33] – C.H. Schmauch, *ISO 9000 for software developers*. Milwaukee: ASQC Quality Press, 1994.

[34] – E. W. Lewis, *Software Testing and Continuous Quality Improvement*. Auerbach Publications,-, 2000.

[35] – D. Galin, *Software Quality Assurance – From Theory to Implementation*. Pearson, -, 2004.

[36] - JUnit - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/JUnit [acedido em Outubro 13, 2010]

[37] - Selenium - Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/Selenium_(software) [acedido em Outubro 13, 2010]

[38] - JMeter - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/JMeter [acedido em Outubro 13, 2010]

[39] - Clover - Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/Clover_(software) [acedido em Outubro 14, 2010]

[40] - Watir - Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/Watir [acedido em Outubro 14, 2010]

[41] - PL/SQL Unit Testing for Oracle – Google Code, [On-line] Disponível em http://code.google.com/p/pluto-test-framework/ [acedido em Outubro 14, 2010]

[42] – Normas ISO – Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/ISO_9000 [acedido em Outubro 17, 2010]

[43]- M. C. Paulk. *Capability maturity model for software – version 1.1.* Technical Report 93-TR-24, CMU/SEI, February 1993.

[44] – M. Fewster, D. Graham, *Software Test Automation*, Addison-Wesley, 1999.

[45] – E. Hendrickson, *The Differences Between Test Automation Success And Failure*, Proceedings of STAR West, 1998.

[46] – B. Marick, *Classic Testing Mistakes*, Proc. of STAR Conference, 1997.

[47] - *Telecommunication Management Forum* – Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/TM_Forum [acedido em Novembro 6, 2010]

[48] – *Telecommunication Management Forum NGOSS* – Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/NGOSS [acedido em Novembro 6, 2010]

[49] - *Telecommunication Management Forum* – TMForum, [On-line] Disponível em http://www.tmforum.org/browse.aspx [acedido em Novembro 6, 2010]

[50] – *Operations Support System* – Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/Operations_support_system [acedido em Outubro 23, 2010]

[51] – Permissões de base de dados GRANT - MSDN [On-line] Disponível em http://msdn.microsoft.com/pt-br/library/ms178569.aspx [acedido em Outubro 23, 2010]

[52] – Schema de base de dados [On-line] Disponível em http://pgdocptbr.sourceforge.net/pg82/sql-createschema.html [acedido em Outubro 23, 2010]

[53] – SOA – Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/Service-oriented_architecture [acedido em Outubro 24, 2010]

[54] - SOAP – Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/SOAP [acedido em Outubro 24, 2010]

[55] - MVC – Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/MVC [acedido em Outubro 24, 2010]

[56] – *Web service* - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/Web_service [acedido em Outubro 27, 2010]

[57] - *Dependency Injection* - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o_de_depend%C3%AAncia [acedido em Outubro 27, 2010]

[58] – POJO – Java Free Forum - [On-line] Disponível em http://javafree.uol.com.br/topic-853529-POJO.html [acedido em Outubro 27, 2010]

[59] - JAR - Wikipedia, [On-line] Disponível em http://pt.wikipedia.org/wiki/Java_Archive [acedido em Outubro 30, 2010]

[60] – DTO – Wikilingue, [On-line] Disponível em http://pt.wikilingue.com/ca/Data_Transfer_Object [acedido em Outubro 30, 2010]

[61] – DAO – Wikipedia, [On-line] Disponível em http://en.wikipedia.org/wiki/Data_access_object [acedido em Outubro 30, 2010]

[62] – SOAPUI - InfoTronicks, [On-line] Disponível em http://infotroniks.blogspot.com/2009/11/comunicacao-com-soap-utilizando-o-soap.html [acedido em Outubro 30, 2010]

Anexos

Anexo 1 – Cenário de utilização da ferramenta TestBeds

Para o funcionamento da ferramenta deve-se criar primeiramente todo o cenário de testes. Este cenário consiste em estruturar o cenário das *TestBeds* no *schema* que será testado, configurar o produto a ser testado e o plano de testes. Os dois últimos passos são feitos através da própria aplicação.

Uma forma de apresentar a ferramenta, passa por criar um pequeno e simples cenário para demonstrar a utilização e os passos a seguir para manipular a ferramenta. O cenário criado passa por carregar um produto para teste, criar um plano de testes e definir os valores de entrada de dados e os resultados esperados de obter aquando da execução do teste. Este cenário permitirá mostrar alguns dos ecrãs da ferramenta.

O primeiro passo passa por escolher a conexão ao *schema* que se deseja testar, inserir os dados de autenticação e carregar no botão *Authenticate* conforme a figura A.1.

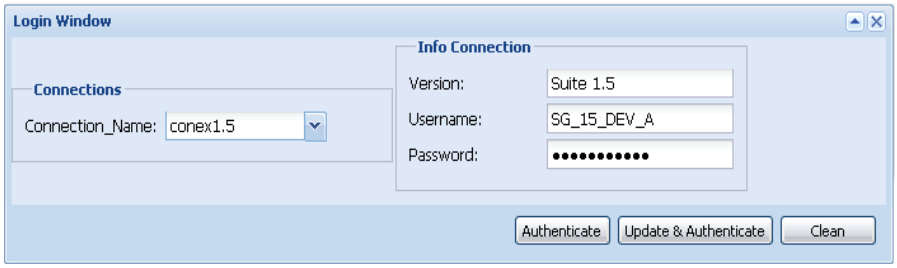

Figura A.1 – Autenticação

O segundo passo é configurar um produto na ferramenta. Este passo tem 3 graus de configuração:

1. Adicionar um produto, ou seja, todas as rotinas de um *schema*;
2. Adicionar apenas um módulo do produto;
3. Adicionar apenas uma rotina pertencente a um módulo.

A figura A.2 apresenta a adição de um produto na sua íntegra.

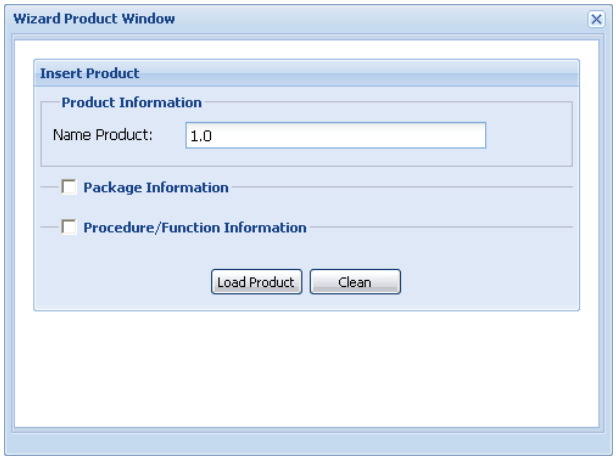

Figura A.2 - Adicionar um produto

O terceiro passo, após adicionar um produto é criar um plano de testes. Os planos de testes são constituídos por vários casos de teste interligados de forma a cobrir o maior número de situações possíveis de falha ou de correcto processamento, simulando assim um ambiente o mais próximo do real. A figura A.3 apresenta a criação de um plano de teste.

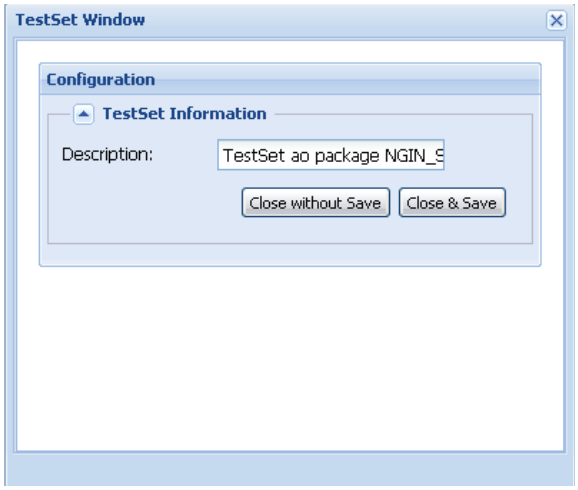

Figura A.3 – Criação de um *TestSet*

Como demonstrado no capítulo 5 na figura 5.1 no modelo de dados um *testset* é constituído por um teste e cada teste pode ter um conjunto de invocações. Portanto, o quarto passo passa por criar um teste e o quinto passo a criação de uma invocação.

Para criar um novo teste existem três possibilidades:

1. Criar um novo teste;
2. Copiar um teste já existente no mesmo *TestSet*;
3. Copiar testes de outro *TestSet*.

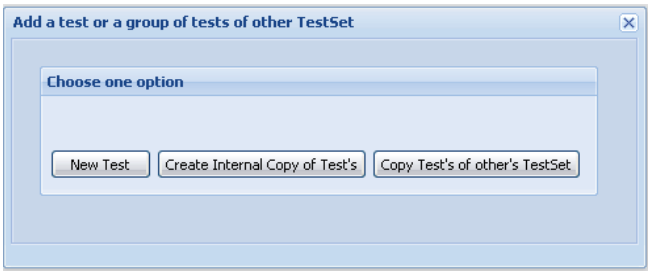

Figura A.4 - Opções de criação de um teste

Neste caso e como o cenário definido passa por criar um teste de raiz a figura apresentada é a de criar um novo teste.

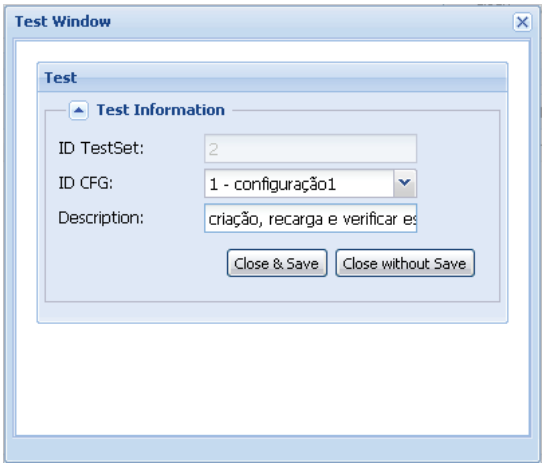

Figura A.5 – Criar um novo teste

Criado um novo teste é criado uma invocação conforme é apresentado na figura A.6.

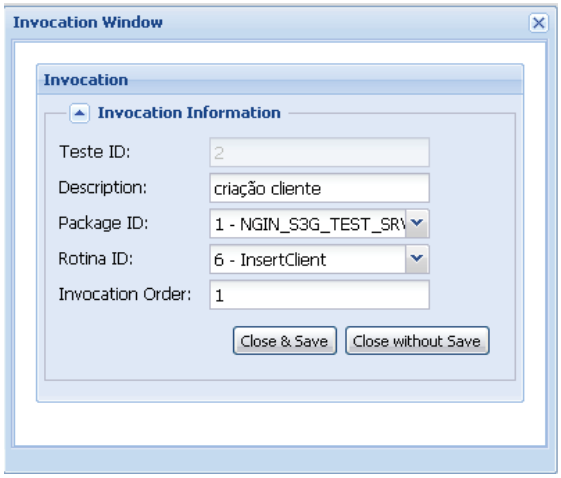

Figura A.6 – Criar uma invocação

O aspecto depois de criado este conjunto de plano de testes, teste e invocação é o apresentado na árvore de navegação da figura A.7.

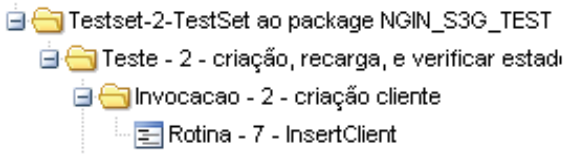

Figura A.7 – Árvore de navegação de testes

Seleccionada na árvore de navegação, figura A.7, a rotina definida na invocação da figura A.8, é possível de visualizar o conjunto de parâmetros que a constituem.

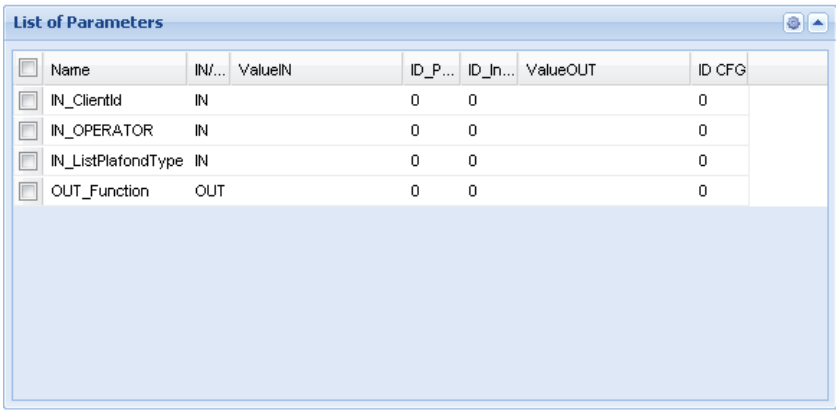

Figura A.8 - Parâmetros da rotina

Para que esteja definido tudo que diz respeito aos planos de testes é necessário definir os valores de entrada da rotina e definir os valores de saída esperados, estes serão o sexto e o sétimo passo respectivamente.

A configuração dos parâmetros de entrada de dados pode ser feita de quatro formas distintas apresentadas a seguir:

1. Utilização do valor definido por omissão do parâmetro;
2. Definir o valor do parâmetro de entrada.
3. Definir que o parâmetro de entrada está associado a um parâmetro de configuração. Este parâmetro herda o valor existente na configuração definida para o teste que a invocação está associada. Esta opção é bastante útil pois permite a reutilização de um valor definido, e caso se altere o parâmetro todos os parâmetros dependentes deste são automaticamente alterados. Este ecrã não é apresentado neste pequeno cenário.
4. Definir que o parâmetro é resultado de um *output* de outra rotina. Este caso pode ser facilmente explicado com um pequeno exemplo. Existindo um teste com duas invocações em que a primeira tem um parâmetro de saída, aquando da execução do teste, a segunda rotina pode ter um parâmetro de entrada que vai herdar o valor do parâmetro de saída da primeira rotina.

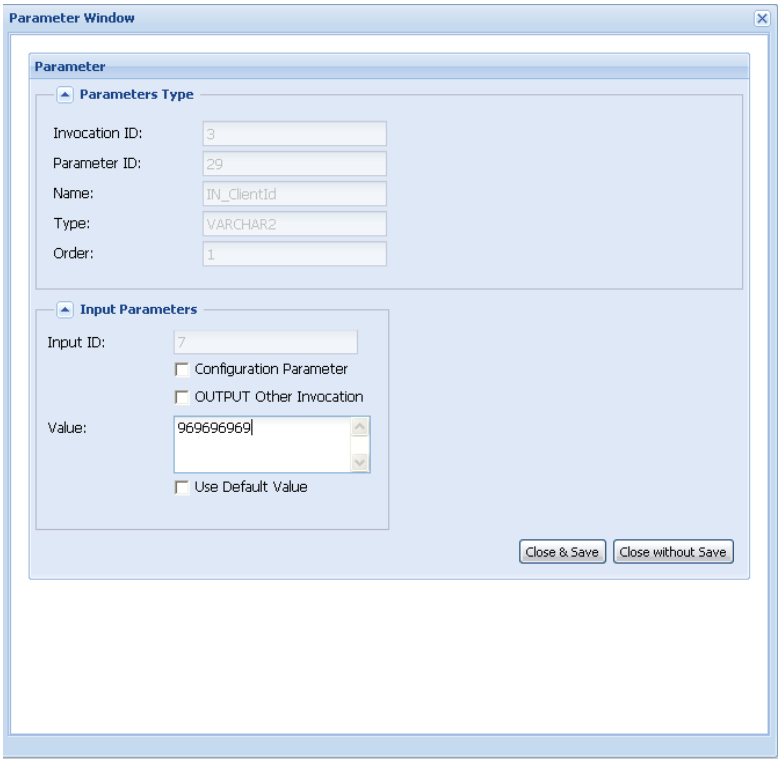

Figura A.9 – Inserção de um parâmetro de entrada

A figura A.9 apresenta a inserção de um valor de um parâmetro de entrada. Na figura A.10 é definido um resultado esperado de uma rotina, o que pode ser feito de 2 formas:

1. Não definir o valor. A mais-valia desta possibilidade prende-se no caso de a solução ser executada no modo "***CheckPoint***" apenas são avaliados com o resultado OK/NOK os parâmetros que interessam e assim o resultado final da invocação da rotina pode ser OK mesmo que um dos parâmetros de saída não apresente o resultado "correcto". Se for executada no modo "***GoldenRun***" apenas são obtidos os resultados dos parâmetros de saída necessários ao utilizador.
2. Definir valor manualmente.

Figura A.10 - Configuração parâmetro de saída com valor definido.

Sempre que é possível fazer a criação de uma entidade, as operações de edição e remoção também estão acessíveis ao utilizador.

Anexo 2 – Cenário de configuração da ferramenta Configuration Manager

Uma forma de apresentar a ferramenta, passa por criar um pequeno e simples cenário para demonstrar a utilização e os passos a seguir para manipular a ferramenta. O cenário escolhido passa por definir uma oferta comercial, sendo essa oferta comercial, a concretização das características parametrizáveis de uma determinada especificação de produto, para disponibilização aos clientes de acordo com um conjunto de variáveis, como sejam: um determinado intervalo de tempo, um número de mensagens, um custo das chamadas, etc. O cenário passa por criar a oferta/parte da oferta comercial Mimo.

Inicialmente é necessário criar uma bolsa de consumo, esta entidade é responsável pela gestão do saldo dos clientes, definindo um conjunto de regras associadas a um determinado valor consumido ou a consumir pelo cliente. Existem bolsas de consumo com diversos tipos de unidade (moeda, volume, evento, tempo), o que permite contabilizar diferentes tipos de consumos. As bolsas de consumo podem ser utilizada por um ou mais produtos.

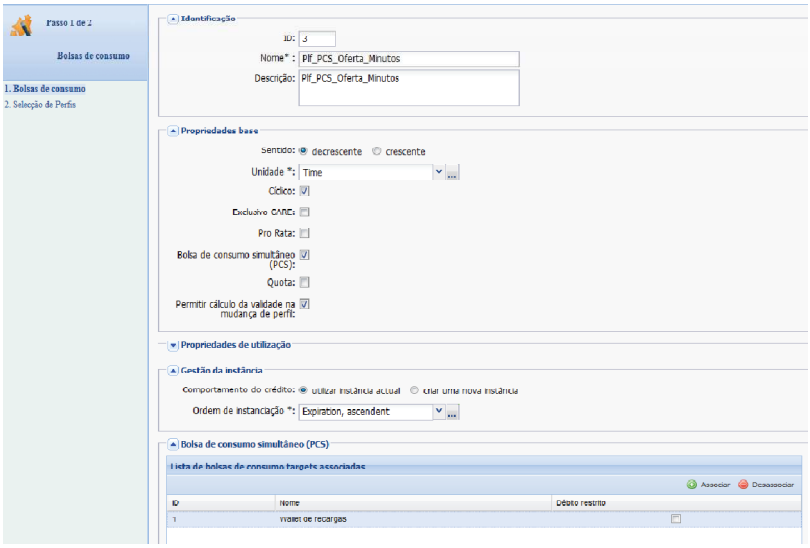

Figura A.11 – Criação de uma bolsa de consumo

Após definida a bolsa de consumo (figura A.12), é necessário associá-la à oferta comercial e definir os parâmetros específicos para essa associação como se pode ver na figura A.13.

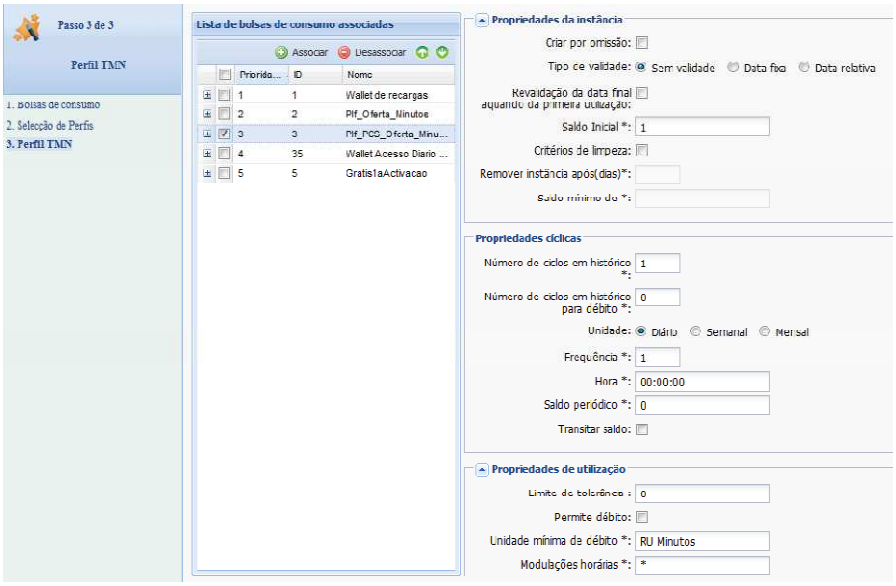

Figura A.12 – Associação da bolsa de consumo à oferta comercial

A seguir é definido um serviço, esta entidade permite o controlo e comercialização individual de um conjunto de características, benefícios ou restrições que definem o seu comportamento. Representa algo a que o cliente tem direito, por ter contratado ou recebido promocionalmente, e que existe sempre no âmbito de utilização de um produto. Exemplos deste tipo de entidade são o serviço de voz, serviço de roaming, etc.

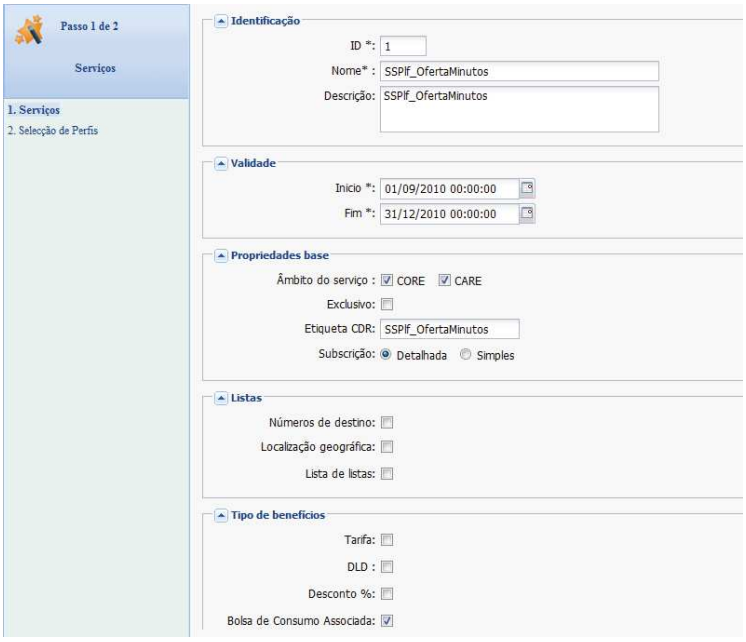

Figura A.13 – Definição de um serviço

Após definido o serviço (figura A.14), é necessário associá-lo à oferta comercial e definir os parâmetros específicos para essa associação como se pode ver na figura A.15.

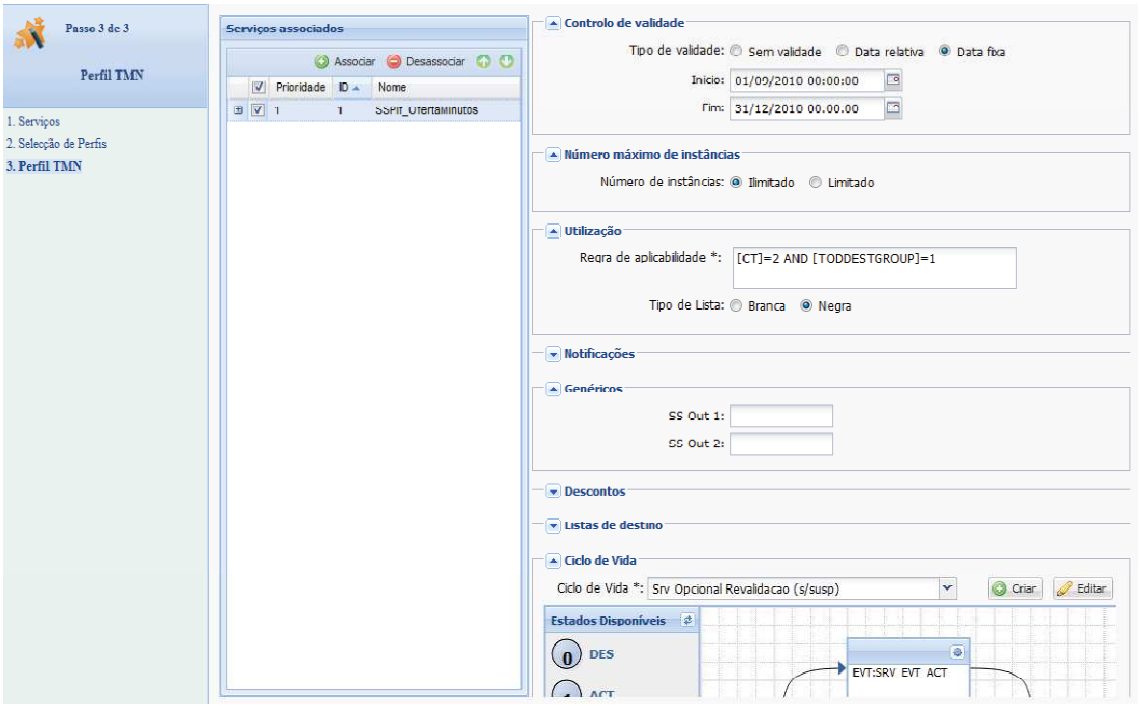

Figura A.14 - Associação do serviço à oferta comercial

Outra entidade que faz sentido definir nesta oferta comercial é um benefício, representando este um "prémio" que pode ser atribuído ao cliente no âmbito de uma promoção ou indexado a uma recarga. Os benefícios podem ser de 2 tipos, crédito, ex 5€ no saldo de chamadas internacionais, e do tipo serviço, ex. desconto nas chamadas ao fim-de-semana. Os benefícios podem ser usados isoladamente ou como conteúdo de um grupo de benefícios.

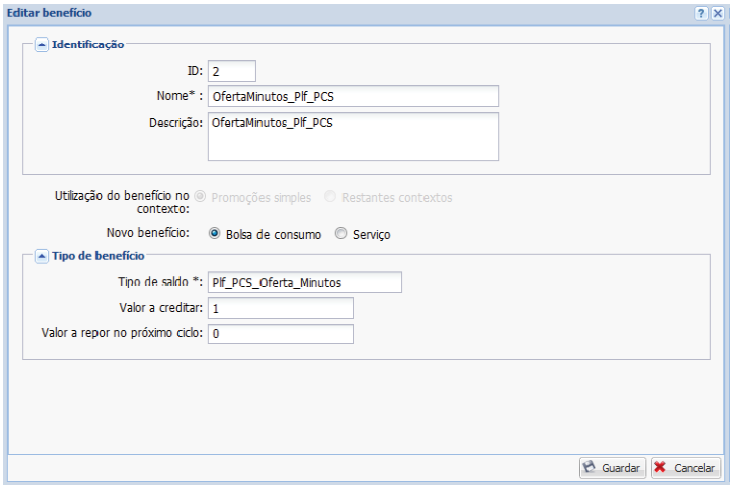

Figura A.15 – Definição de um benefício

Como foi dito atrás um benefício (figura A.15) pode ser atribuído numa promoção, sendo que esta entidade existe no sistema para modelar acções de *marketing*. A sua configuração permite definir a atribuição de benefícios ao cliente, num determinado período de tempo e mediante regras baseadas na ocorrência de determinado evento. A configuração de uma promoção é apresentada na figura A.16.

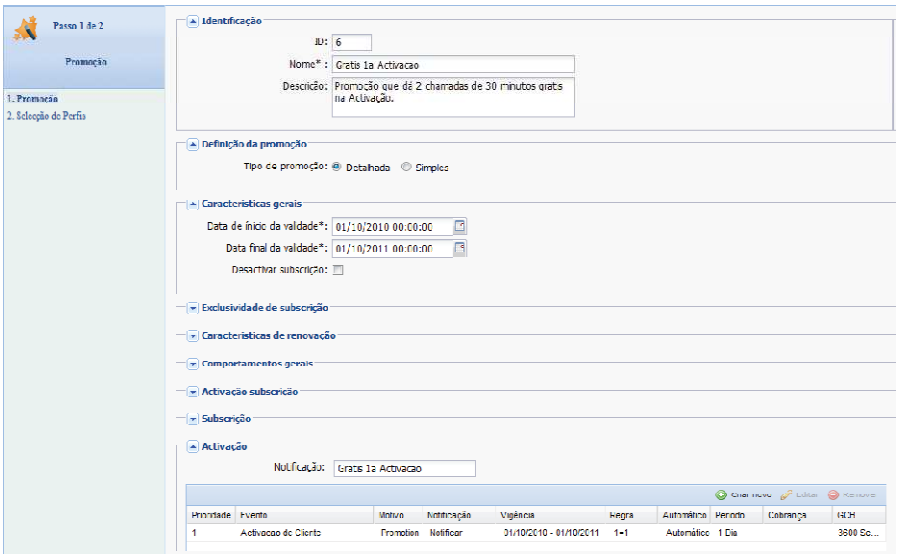

Figura A.16 – Definição de uma promoção

Este pequeno cenário permite perceber parte da estrutura e do aspecto gráfico da ferramenta *Configuration Manager*. Muitas outras entidades são necessárias para a definição de uma oferta comercial real, alarmes, notificações, recargas, ciclos de vida, e muitas mais, sendo o *Configuration Manager* a ferramenta usada para configurar todas essas entidades, formando assim um catálogo de produtos/serviços que a operadora tem para disponibilizar aos seus clientes.