Departamento de Engenharia Informática

Relatório de Estágio

Mestrado em Engenharia Informática - Computação Móvel

SAFER Response

Joel Rodrigues Fernandes

Outubro de 2013

Departamento de Engenharia Informática

Relatório de Estágio

Mestrado em Engenharia Informática - Computação Móvel

SAFER Response

Joel Rodrigues Fernandes

Estágio de Mestrado realizado sob a orientação do Doutor Carlos Fernando Almeida Grilo, Professor da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria.

Outubro de 2013

À Minha Família

Agradecimentos

Nesta página aproveito para agradecer às pessoas que tornaram possível chegar onde cheguei: a finalização de uma nova etapa nos meus estudos.

Em primeiro lugar, o meu muito obrigado ao Professor Carlos Grilo, pelas horas disponibilizadas e pelo apoio prestado durante este ano.

Quero agradecer ao António Marcelo, da Tecmic, por me indicar este projeto tão interessante e pelo apoio prestado. Também da Tecmic um meu muito obrigado ao David Mendes, ao João Amaro e ao João Ribeiro por me fornecerem guarida quando precisei e também pelo apoio no trabalho.

Apesar de ter tido uma entrada tardia no projeto, o meu muito obrigado ao André Leal, pelos testes ao sistema, correções apresentadas e ajuda no final da implementação do projeto.

Um muito (mas mesmo muito!) obrigado aos papás que "patrocinaram" a minha escolaridade, e pela paciência que tiveram quando disse que não podia ir a casa em alguns fins-de-semana. Obrigado Manuel e Isaura Fernandes.

Em maré de agradecimentos, fica aqui o meu muito obrigado ao meus colegas de curso e de tuna André Sousa e David Carvalhana, por ideias que não têm fim e por aqueles bons momentos que fizeram esquecer algum dia pior.

Por último, mas não menos importante, o meu muito obrigado à **Sarah Lopes**. Obrigado pelo apoio que nunca faltou, e pela compreensão por alguma coisa que tenha corrido menos bem.

Resumo

Este documento visa dar a conhecer o trabalho realizado no âmbito do projeto SAFER Response. O projeto foi proposto pela empresa Tecmic S.A. e insere-se no Mestrado em Engenharia Informática – Computação Móvel da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria.

O SAFER (System for Advanced FiEld opeRation) Response é um projeto que se enquadra na área de gestão de emergências, uma área especializada derivada da gestão de ocorrências. A gestão de emergências trata do processo de acompanhamento e gestão de situações que possam ocorrer tais como incêndios, cheias, sismos, derrames, etc. Como se pode antever, é um sistema destinado a corporações como bombeiros, forças de segurança, proteção civil, etc. A Tecmic S.A. possui um sistema proprietário de gestão de emergências, denominado XTraN 4 Forces. Este sistema dá origem ao projeto SAFER, cujo objetivo é a expansão do XTraN 4 Forces de modo a abranger mais entidades, através da construção de um módulo para suportar a lógica de gestão relacionada com as forças de segurança.

O trabalho realizado divide-se em duas componentes técnicas: uma componente de servidor e uma componente cliente *desktop*.

A componente de servidor consiste na criação de um servidor que disponibiliza um conjunto de *webservices* de acesso e controlo de dados relacionados com Forças de Segurança. Estes dados são geridos pela componente *desktop*, uma aplicação desenvolvida na forma de *plugin*, de modo a alargar o sistema existente XTraN 4 Forces, permitindo gerir a informação da componente de servidor.

Palavras-chave: Gestão de Emergências, Gestão de Ocorrências, Forças de Segurança.

Abstract

This document describes project SAFER Response. The work reflected in this document has occurred during internship at Tecmic SA. This internship is also part of Computer Science – Mobile Computing master degree at Escola Superior de Tecnologia e Gestão of Instituto Politécnico de Leiria.

SAFER – System for Advanced FiEld opeRation Response – is a project introduced by Tecmic. This project aims at extending an existing project, related to Emergency Management, named XTraN 4 Forces. This area can be thought of a specialization of occurrence management area. Emergency management handles situations which are disastrous by nature, such as flooding, wildfires, earthquakes, atmospheric dispersions, etc. From the enumerated items, one can figure the target users of such a project: firemen, security forces officers, civil protection officers, etc.

XTraN 4 Forces was developed with civil protection case study in mind. SAFER Response was born from XTraN 4 Forces in order to extend it: handle security forces logic.

By further developing the system, not only will XTraN 4 Forces grow with more core functionality, but the system can now be handled by civil protection and security forces users. The goal is to have both core groups of users being able to use the system to the best of their advantage.

In order to achieve this kind of functionality, SAFER Response consists of a plugin application, which can be added specifically for security forces users. This plugin approach has the advantage of leaving core system, as well as any already implemented features unaffected. SAFER project is composed of two different applications: server and client.

Server application provides webservices which allow client applications to handle data centrally. These webservices serve as main data source for the client application – which works as a plugin for XTraN 4 Forces.

Key-Words: Emergency management, occurrence management, security forces.

Índice de Figuras

Índice de Quadros

Lista de Siglas

Este documento faz uso frequente de siglas cujo significado se apresenta na tabela abaixo. Estas siglas são tipicamente apresentadas aquando da sua primeira utilização:

Acrónimo	Significado
4 Forces	XTraN 4 Forces
TECMIC S.A.	Tecnologias de Microeletrónica, S.A.
FS	Forças de Segurança
SIGEL	Sistema Integrado de Gestão de Emergência e Logística
ANPC	Autoridade Nacional de Proteção Civil
SAFER	System for Advanced FiEld opeRation
DCT	Defesa Civil do Território
NATO	North Atlantic Treaty Organization
ONDCT	Organização Nacional da Defesa Civil do Território
SNPC	Serviço Nacional de Proteção Civil
SNB	Serviço Nacional de Bombeiros
CEFF	Comissão Especializada em Fogos Florestais
SNBPC	Serviço Nacional de Bombeiros e Proteção Civil
INEM	Instituto Nacional de Emergência Médica
MOHID	Modelo Hidrodinâmico
KML	Keyhole Markup Language
LiDAR	Light Detection And Ranging
PSP	Polícia de Segurança Pública
SADO	Sistema de Apoio à Decisão Operacional
PCGO	Proteção Civil Gestão de Ocorrências
WCF	Windows Communication Foundation
CISEM	Centro Integrado de Segurança e Emergências de Madrid

MSDE	Microsoft SQL Server Data Engine
RMS	Records Management System
CAD	Computer Aided Dispatch
GIS	Geographic Information System
API	Application Programming Interfaces
EEM	Empresa da Eletricidade da Madeira
IDE	Integrated Development Environment
CLR	Common Language Runtime
POCO	Plain Old CLR Objects
ORM	Object-Relational Mapping
REST	REpresentational state transfer
SOAP	Simple Object Access Protocol
XML	eXtensible Markup Language
HTTP	HyperText Transfer Protocol
MVC	Model View Control
DAL	Data Access Layer
JSON	JavaScript Object Notation
DBMS	Database Management System
SQL	Structured Query Language
FNH	Fluent NHibernate
CRUD	Create, Replace, Update e Delete
DI	Dependency Injection
IoC	Inversion of Control
UoW	Unit of Work
LINQ	Language-Integrated Query
CAOP	Carta Administrativa Oficial de Portugal
UI	User Interface
HTTPS	HyperText Transfer Protocol Secure
DLL	Dynamic-link Library
VS	Visual Studio
URL	Uniform Resource Locator
URI	Unique Resource Identifier

Índice

1 Introdução

A Gestão de Emergências é uma área de gestão vocacionada para a otimização de recursos e minimização dos impactos aquando da ocorrência de uma situação perigosa que coloque em risco o bem-estar ou a saúde, individual ou pública.

O XTraN 4 Forces – ou simplesmente 4 Forces – é um sistema proprietário da empresa Tecmic S.A. [1] que se enquadra na área de Gestão de Emergências, mais especificamente, de catástrofes naturais. O sistema permite dar resposta à evolução de ameaças causadas pelas quatro forças naturais, fogo, água, ar e terra, e facilita a tomada de decisões através de simulações, representações gráficas e gestão de recursos em tempo real. Desta forma é possível obter uma otimização da atribuição e utilização de recursos e uma minimização do impacto das catástrofes através de uma melhor tomada de decisões.

Como qualquer sistema informático, o XTraN 4 Forces encontra-se em constante evolução, de forma a melhorar a usabilidade e oferecer melhores funcionalidades que permitam facilitar o processo de gestão das emergências com o menor custo possível. O projeto SAFER (System for Advanced FiEld operation) Response visa expandir o XTraN 4 Forces de forma a suportar as necessidades das Forças de Segurança (FS).

1.1 Entidade de Acolhimento

A entidade de acolhimento do estágio foi a Tecmic S.A. - Tecnologias de Microeletrónica, S.A.. Fundada em 1988, esta multinacional portuguesa conta em 2013 com cerca de quarenta colaboradores. Sendo a área principal de negócio a gestão de frotas, a empresa possui também soluções em áreas complementares de negócio, desde gestão de logística, gestão de recursos remotos, gestão de emergências, passando pela área da saúde, entre outras.

No que diz respeito a parceiros, a Tecmic conta com entidades como Vodafone, Motorola, Tele Atlas, Link e INOV, etc. A área de influência da Tecmic estende-se a vários países como Espanha, Suíça, Alemanha, Áustria, Irlanda, Marrocos, Angola e, mais recentemente, o

Brasil. São vários os clientes das soluções Tecmic, de entre os quais se destacam empresas como Alliance Healthcare, Carris, Correios de Portugal (CTT), Luís Simões, Portugal Telecom e Unilever. Para além de empresas destacam-se instituições como Assembleia da República, Automóvel Club de Portugal (ACP), Banco de Portugal e Exército Português. A tabela abaixo enumera as principais soluções da Tecmic, das quais faz parte o sistema XTraN 4 Forces.

Tabela 1 - Gama de produtos Tecmic S.A.

Designação do Produto	Área de aplicação
XTraN	Gestão de Frotas
iZi TraN	Gestão de Frotas *Online*
XTraN 4 Forces	Gestão de Emergências
Siga	Gestão de Acessos
Simor	Monitorização Remota
Ecogest	Gestão de Ecopontos
ASIC's	Circuitos Integrados

1.2 Aplicação XTraN 4 Forces

O sistema XTraN 4 Forces (ou simplesmente 4 Forces) é uma evolução de um outro sistema denominado SIGEL – Sistema Integrado de Gestão de Emergência e Logística. Etimologicamente, XTraN 4 Forces provém do sistema XTraN, a solução Tecmic para gestão de frotas. 4 Forces provém das quatro forças naturais que compõem a natureza: fogo, água, ar e terra.

O 4 Forces é, segundo palavras da Tecmic, "a aplicação de Gestão de Emergências que permite realizar de forma rápida e efetiva a tomada de decisões para o combate das catástrofes naturais como incêndios, cheias, descargas atmosféricas, derrames e tremores de terra". É um sistema que tem potencial para ser utilizado por um conjunto de entidades que necessitem de lidar com estas situações, incluindo bombeiros, socorristas, exército, proteção civil, marinha, forças de segurança, etc.

O fator comum que une este grupo de entidades como potenciais utilizadores do sistema prende-se com a gestão de recursos no terreno, mais concretamente, agentes operacionais. Estes agentes não podem exercer as suas funções de forma ininterrupta, necessitando de atenção por parte de quem gere os recursos. Da mesma forma que o combustível numa viatura

não é infinito, os agentes necessitam de pausas para comer e descansar. É função do 4 Forces facilitar esta gestão para que os recursos no terreno estejam disponíveis da forma o mais eficiente possível, evitando a duplicação de agentes no terreno por parte de diferentes entidades, a permanência dos recursos no terreno por tempo superior ao desejável e permitindo a rotatividade dos agentes e uma melhor aproximação e combate aos problemas nos momentos cruciais.

No que toca às ameaças, o sistema consegue prever a evolução das mesmas através de técnicas computacionais que envolvem dados cartográficos (mapas, modelos de combustível, etc.) e algoritmos conhecidos de modelação computacional da evolução de incêndios, cheias, correntes marítimas, dispersões atmosféricas, etc. A modelação pode envolver ainda informação proveniente de sistemas externos, como a obtenção da previsão de condições meteorológicas para um determinado intervalo de tempo ou ainda dados que sejam modelados por sistemas de terceiros. Todos os cálculos são afetos a um intervalo de tempo que é definido pelos utilizadores do sistema. Uma vez realizados os cálculos, é possível gerar resultados visíveis em mapas bi e tridimensionais.

Contando ainda com uma componente de relatórios e *dashboards*, é possível avaliar a efetividade da gestão dos recursos e obter resultados mensuráveis. As funcionalidades base do sistema encontram-se resumidas na tabela seguinte:

Tabela 2 - Funcionalidades resumidas do sistema 4 Forces.

Grupo	Categoria
Gestão	Emergências
	Ocorrências
	Recursos
	Dashboards informativos
Simulações	Incêndios
	Cheias
	Dispersões atmosféricas
	Derrames marítimos
	Localização de corpos
Previsões	Tempo até à próxima refeição/período de descanso de agentes
	Quilómetros que um veículo pode efetuar
Tempo real	Condições meteorológicas
	Vídeo vigilância florestal – Ciclope
	Localização dos meios

No presente estado, o 4 Forces encontra-se em fase de teste, customizado para ser utilizado

pela ANPC - Autoridade Nacional de Proteção Civil. O objetivo é expandir o sistema a outras entidades e neste relatório é descrita a expansão realizada para que o sistema possa ser utilizado pelas Forças de Segurança. É neste contexto que surge o SAFER Response.

1.3 SAFER Response

O projeto SAFER - System for Advanced FiEld opeRation – Response nasce com o objetivo de estender o 4 Forces de forma a incluir um novo grupo de utilizadores: as Forças de Segurança (FS).

À semelhança da ANPC, e de todas as entidades que venham a utilizar o sistema, as funcionalidades base, de um modo simplista, enquadram-se na lista ilustrada na Tabela 2. Salvaguardam-se as personalizações para o caso específico da ANPC, que servem lógica específica e não são reutilizáveis diretamente por mais nenhuma entidade que utilize o sistema. Este é um dos desafios maiores da construção do sistema, a capacidade de servir a vários clientes um conjunto de funcionalidades comuns sem prejuízo das necessidades específicas de cada entidade.

Em reuniões com elementos das FS, foi apurada a necessidade da utilização do sistema no dia-a-dia. Sendo o 4 Forces um sistema destinado à gestão de emergências, que no caso das FS se traduzem no contexto de ocorrências e missões, faz sentido que o sistema possa gerir também as mesmas ocorrências e missões, centralizando a gestão como um todo na mesma aplicação.

O SAFER vem, assim, transformar o 4 Forces num sistema capaz de gerir a atividade diária das FS como ocorrências e missões quotidianas que, quando se tratem de emergências – cheias, incêndios, etc., podem ser geridas com auxílio ao arsenal de técnicas de modelação que o 4 Forces já possui (como previsões, simulações, visualização em tempo real, etc.). O SAFER faz com que a utilização deste sistema deixe de ser apenas em casos excecionais de emergência para ser um sistema cuja utilização é diária e de forma praticamente ininterrupta.

Com esta nova necessidade surgem novas responsabilidades e complicações que não existiam com a ANPC, ou pelo menos não de forma tão evidente e crucial. Surgem então questões de segurança, integração, alarmística, relatórios customizados, etc.

Este projeto consiste numa componente *desktop* e numa componente *web*. A componente *desktop* consiste no desenvolvimento de um *plugin* que se destina a ser integrado na aplicação 4 Forces e cuja função é a capacidade de gerir toda a informação das FS, desde informação geral respetiva à esquadra, até ocorrências, missões, agentes, equipas, equipamentos e

veículos. O sistema deve ser capaz de identificar a localização dos agentes e veículos em qualquer instante e mostrar essa informação de uma forma simples e intuitiva num mapa que deve ainda conter, entre outros, a localização de ocorrências, missões e pontos de interesse. A capacidade de visualização do histórico de envolvência de agentes, utilização de equipamentos e evolução de ocorrências e missões também é um requisito de elevada importância, pois permite tirar conclusões e auxiliar a tomada de decisões futuras, através da geração de relatórios personalizados.

A componente web disponibiliza *webservices* para serem acedidos por aplicações cliente. Esta componente está pensada não só para servir o *plugin* mas também para, mais tarde, servir aplicações móveis que irão surgir neste projeto. Deste modo o sistema é arquiteturado desde cedo para ser orientado a serviços, de modo a ser escalável para o futuro. A componente *web* deve ser capaz de gerir as mesmas entidades que a aplicação cliente de forma a atuar como fonte de dados centralizada.

1.4 Motivação e objetivos

As emergências são situações que, por natureza, têm de ser combatidas o mais rapidamente possível. São situações que se podem antecipar mas não se podem prever com data exata ou impacto determinado. É possível prever um aumento do número de incêndios em determinadas alturas do ano, mas não é possível prever a data exata em que irá ocorrer um incêndio, qual a área mais afetada ou quais os danos que irá causar. Mas, quando acontecer, toda a ajuda é preciosa. No entanto, é impossível estar em todo o lado ao mesmo tempo e é necessário organizar os recursos para onde a atenção é mais necessária em determinado momento ou para onde pode causar maior efeito no combate ao problema. É crucial, para que não haja recursos em falta, áreas não atendidas, recursos duplicados no terreno – provenientes de instituições diferentes – etc., que haja uma gestão eficaz de recursos e da própria emergência em si.

O facto de se tratar de forças de segurança motiva ainda mais o trabalho a desenvolver. A gestão de ocorrências e missões das FS complementa a gestão de emergências, aumentando a produtividade e o automatismo do processo de gestão, libertando os recursos para o que realmente importa: ajudar a manter a ordem.

O objetivo do SAFER é providenciar às FS uma ferramenta que permita uma gestão eficaz dos processos de negócio capaz de automatizar os processos que são hoje em dia realizados em papel ou em sistemas próprios, não centralizados. Decisões mal tomadas podem ser

muitas vezes fatais e quando se fala de emergências, tempo não é apenas dinheiro. Muitas vezes são vidas humanas em jogo, o que só por si faz da gestão de emergências uma área na qual é necessário apostar. Melhorar a tomada de decisões e o impacto das decisões tomadas para que os agentes envolvidos possam concentrar-se mais naquilo que realmente é necessário: salvar vidas, bens e manter a ordem. É esta a motivação para o desenvolvimento deste sistema.

1.5 Estrutura do Documento

Esta secção descreve a organização deste documento. Os restantes capítulos deste relatório estão organizados do seguinte modo:

- Capítulo 2: Neste capítulo é abordado o estado atual das áreas afetas ao sistema, incluindo a ANPC, as FS, o XTraN 4 Forces, assim como sistemas relevantes existentes no mercado atualmente.
- Capítulo 3: Aqui encontra-se descrita a metodologia utilizada para o desenvolvimento do projeto, assim como o planeamento e gestão de projeto.
- Capítulo 4: Este capítulo descreve em pormenor o desenvolvimento realizado, começando por explicar as opções tecnológicas e a arquitetura do sistema existente, passando para o modelo de domínio e implementação das aplicações servidora e cliente.
- Capítulo 5: neste capítulo são discutidos resultados e são tecidas considerações a ter em conta em trabalho futuro.

2 Estado da arte

Este capítulo visa dar a conhecer as áreas abrangidas pelo projeto: gestão de emergências, a importância do 4 Forces, sistemas atualmente utilizados e outros sistemas relevantes.
Por ordem de conteúdos, o capítulo começa por introduzir a área de gestão de emergências. Segue-se uma pequena análise da ANPC e do processo de gestão de emergências utilizado, seguindo-se a explicação do 4 Forces no contexto da APNC. O capítulo continua com a introdução das Forças de Segurança. Finalmente, são descritos brevemente outros sistemas relevantes nas áreas de gestão de emergências e Forças de Segurança.

2.1 Gestão de emergências

Aquilo a que chamamos hoje de gestão de emergências, foi até há poucas décadas conhecido simplesmente como "defesa civil", tendo posteriormente evoluído para "proteção civil" ou "gestão de emergências". A expressão "proteção civil" é utilizada mais vulgarmente para designar o organismo que gere as emergências, sendo a expressão "gestão de emergências" mais utilizada para referir a área em si mesma.
A defesa civil originalmente consistia em defender o território em situação de guerra contra invasores e a responsabilidade para o efeito era delegada aos exércitos assim como aos seus chefes e reis. Catástrofes naturais como incêndios, sismos, cheias, etc. não eram alvo de tão grande importância numa fase primordial porque eram consideradas, citando Freitas, C.M. e Gomez, C.M. [2], "manifestações da providência divina", difíceis de prever. Mas com a estabilização das comunidades em centros populacionais de dimensão crescente tornou-se importante, por razões estratégico-económicas, defender esses mesmos centros populacionais em caso de emergência, seja esta de natureza militar ou natural (catastrófica).
Em Portugal, o emprego do conceito de defesa civil remonta à altura da Segunda Guerra Mundial [3], altura em que a Defesa Civil do Território (DCT) foi criada com responsabilidade de defesa do país em caso de guerra [4]. A DCT foi reestruturada com a

entrada de Portugal na NATO (*North Atlantic Treaty Organization*), originando a Organização Nacional da Defesa Civil do Território (ONDCT). Em 1974 a ONDCT deixou de existir e em 1975 foi criado o Serviço Nacional de Proteção Civil (SNPC). É nesta fase que a designação de defesa civil passa a dar lugar à designação de proteção civil, em que o âmbito de ação do organismo deixa de ser militar para ser civil. Posteriormente foram criados o Serviço Nacional de Bombeiros (SNB) e a Comissão Especializada em Fogos Florestais (CEFF). Finalmente, o SNPC, o SNB e a CEFF são fundidos no Serviço Nacional de Bombeiros e Proteção Civil (SNBPC), que mais tarde se passou a designar Autoridade Nacional de Proteção Civil. A responsabilidade máxima da ANPC está em última análise afeta ao Primeiro-Ministro [5].

A gestão de emergências refere-se à atividade realizada (não estritamente) pela ANPC, que se destina à prevenção de situações de emergência, ou atuação em caso de catástrofe eminente de forma a reduzir a perda de vidas e de bens materiais. A proteção civil é, em palavras citadas do sítio da ANPC, "a atividade desenvolvida pelo Estado, Regiões Autónomas e Autarquias Locais, pelos cidadãos e por todas as entidades públicas e privadas, com a finalidade de prevenir riscos coletivos inerentes a situações de acidente grave ou catástrofe, de atenuar os seus efeitos, proteger e socorrer as pessoas e bens em perigo quando aquelas situações ocorram." [6].

A ANPC dispõe de um leque alargado de competências de atuação por terra, água ou ar, desde salvamentos, policiamento até mesmo à saúde. Este leque de competências é assegurado em terra pelas forças de segurança (FS), Forças Armadas, bombeiros e sapadores florestais. As operações marítimas estão a cargo da Autoridade Marítima Nacional e as operações aéreas estão atribuídas à Autoridade Aeronáutica Nacional. Transversalmente, existem ainda serviços de saúde, entre os quais o Instituto Nacional de Emergência Médica (INEM). As entidades suprarreferidas cooperam entre si em caso de catástrofe de forma a garantir o combate à mesma da melhor forma possível, estando a ANPC responsável por mediar a cooperação. No que diz respeito à gestão de emergências, a ANPC atua de acordo com o ciclo de gestão de emergências [7]–[9], que é apresentado na Figura 1.

A disposição das quatro etapas na imagem tem em conta as fases pré e pós catástrofe [10] que se encontram em cima e em baixo, respetivamente. A secção pré catástrofe é composta pelas etapas de mitigação e preparação. Estas etapas não estão dependentes da ocorrência de uma catástrofe, podendo (e devendo) ser realizadas de forma contínua. A fase da mitigação é orientada essencialmente à redução do impacto de emergências, em que a ANPC realiza levantamentos de risco de possíveis áreas vulneráveis e cria planos [6]. Em conjunto com os

levantamentos de risco são efetuadas previsões e a avaliação de danos associados aos mesmos, são contabilizados os recursos disponíveis para atuar em caso de necessidade e quais é que são mais facilmente alocáveis em caso de catástrofe. Em cada previsão é tido em conta o pior cenário possível e o planeamento de soluções de emergência é realizado de acordo com os levantamentos efetuados.

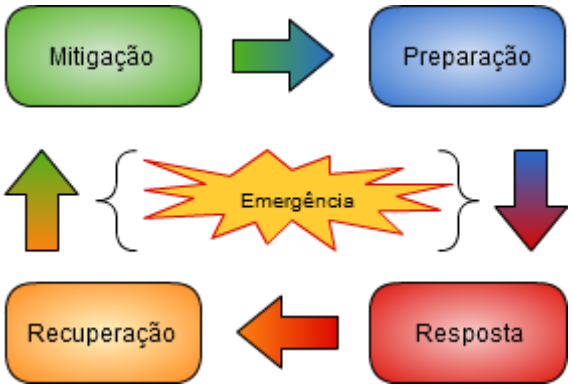


Figura 1- Ciclo de gestão de emergências.

No entanto, apenas a existência de planos não é suficiente. É necessário garantir que esses planos têm condições para ser aplicados da melhor forma possível, divulgando-os, executando simulações, ações de formação (a pessoal especializado, escolas, etc.), verificação constante dos equipamentos, etc. Tais ações são efetuadas na etapa de preparação.

A ocorrência de uma catástrofe despoleta a etapa de resposta. Tendo em conta os planos realizados na etapa de mitigação, são mobilizados os recursos com o fim de proteger pessoas e bens de forma a minimizar as perdas. Os bens mobilizados podem ser provenientes de uma só entidade mas, na maioria dos casos, são provenientes de várias entidades. Exemplo típico será um incêndio de média ou grande dimensão, que mobiliza agentes de corpos de bombeiros ou sapadores (pela razão óbvia do combate às chamas), INEM ou outros socorristas para a eventualidade da necessidade de prestação de assistência médica imediata e, ainda, forças de segurança para averiguar as causas, realizar o corte de trânsito, etc.

Resolvida a ocorrência, dá-se início ao processo de recuperação. É também da responsabilidade da ANPC o apoio pós catástrofe com vista a apoiar a restauração da normalidade nas zonas afetadas. Retomando o caso do incêndio, são exemplos de ações tomadas referentes à etapa de recuperação a reflorestação de zonas ardidas, a reconstrução de edifícios ou a replantação de colheitas.

Como é possível observar na imagem acima, o processo de gestão de emergências é um processo iterativo que não tem um fim determinado. Em cada nova emergência gerida é obtido novo conhecimento que é depois tido em conta na mitigação e prevenção de futuras emergências. Desta forma, possíveis falhas que tenham ocorrido podem ser colmatadas para

que não voltem a acontecer.

2.2 O 4 Forces na ANPC

O SIGEL, posteriormente rebatizado como 4 Forces, visa auxiliar a ANPC nesta tarefa importante que é a gestão de emergências. O seu foco é de caráter reativo e surge com a finalidade de auxiliar a resolução de emergências. No diagrama do ciclo de gestão de emergências, as funções do 4 Forces enquadram-se maioritariamente na etapa da resposta, fase em que toda a ajuda é escassa, e que a utilização do 4 Forces visa auxiliar.

Desde incêndios, cheias, descargas de corpos até dispersões atmosféricas, o 4 Forces está munido de múltiplas técnicas de simulação. As simulações, depois de executadas, podem ser visualizadas em mapas bidimensionais e tridimensionais, providenciando de forma fácil e intuitiva ao utilizador (agente ou responsável) informação acerca das possíveis evoluções de determinada emergência. No estado atual, o 4 Forces conta com as seguintes simulações:

Tabela 3 - Simulações disponíveis no 4 Forces.

Elemento natural	Tipo de simulação
Ar	Dispersão Atmosférica
Fogo	Modelo em Cone
	Modelo 3%
	BlueFire
Água	Cheia fluvial
	Cheia urbana
	Derrame petrolífero
	Localização de corpos

As simulações apresentadas na tabela anterior foram desenvolvidas por empresas externas à Tecmic: a BlueCape [11] e a Action Modulers [12]. Da autoria da BlueCape são as implementações das simulações de dispersões atmosféricas, – baseadas na modelação CALPUFF [13], [14] – e evolução de incêndios – através de algoritmos que têm em conta aspetos como a velocidade do vento e a topografia do terreno (modelos de combustível e ocupação do território). Da Action Modulers resultaram as simulações de cheias (urbanas e fluviais), localização de corpos humanos na água e derrames petrolíferos, utilizando como base o Modelo Hidrodinâmico (MOHID). É também da autoria da Action Modulers a implementação de um *plugin* de Simulações Virtuais, que será abordado mais tarde neste

documento.

Para "alimentar" as simulações, é necessário obter dados meteorológicos com frequência. Para o efeito, existe um serviço que é instalado na mesma máquina onde é instalado o 4 Forces, que obtém informação meteorológica atualizada periodicamente. Esta informação é importantíssima para garantir que o sistema realiza as simulações de forma tão precisa quanto possível.

Uma vez executadas as simulações, os resultados são apresentados num mapa, existindo um *plugin* de controlo temporal que permite a visualização dos resultados em função de um intervalo de tempo especificado pelo utilizador. A título de exemplo, se considerarmos uma determinada simulação para um intervalo de tempo – digamos, 3 horas – é possível visualizar os resultados da simulação para qualquer instante desse mesmo intervalo. É possível, assim, acompanhar a evolução da simulação ao longo do tempo, uma vez que os resultados das simulações estão diretamente relacionados com o tempo.

Existe ainda um *plugin* que permite exportar os resultados das simulações para o formato Keyhole Markup Language (KML), de modo a serem utilizáveis pelo *software* Google Earth [15]. Este último *plugin* integra ainda o próprio Google Earth numa janela do 4 Forces e, como tal, permite ainda carregar os resultados exportados e visualizar os mesmos em mapas 3D. Mais uma vez, com o auxílio do *plugin* de controlo temporal, é possível recriar a evolução da simulação e, ainda, pré-visualizar os dados para um determinado intervalo de tempo. Ao definir intervalos de tempo muito reduzidos, é possível visualizar a informação de forma animada, como se de um vídeo se tratasse.

Para além de todas as simulações apresentadas, o sistema integra também o sistema de videovigilância florestal CICLOPE [16]–[18], desenvolvido pela empresa INOV [19], de uma forma pioneira em Portugal, permitindo observar em tempo real o estado de zonas florestais onde os equipamentos se encontram montados. Uma vez que os equipamentos se encontram montados em torres de vigilância situadas em locais estratégicos de elevada altitude e possuem capacidade de visão 360º através de câmaras rotativas, é possível observar uma área de vários quilómetros com uma só torre. As imagens podem ser obtidas em tempo real através de uma ligação à *internet*, podendo ser também enviadas ordens de volta para os equipamentos (enviar pedidos para que as câmaras rodem, realizar rotinas de manutenção, etc.). Além da capacidade de visualização em tempo real das imagens, as próprias torres CICLOPE encontram-se munidas com capacidade de deteção automática de incêndios, através da medição de infravermelhos [20], [21] ou da utilização de técnicas *Light Detection And Ranging (*LiDAR) [22], [23]. Desta forma, conseguem detetar-se incêndios mesmo que

não haja uma pessoa a olhar para as imagens. Sendo as torres auto suficientes em termos energéticos, através de baterias e painéis solares, este sistema é uma solução interessante para a monitorização de incêndios em Portugal [18].Estas simulações e integração com o sistema CICLOPE constituem as funcionalidades que formam o "núcleo" do 4 Forces.

O sistema permite ainda gerir entidades relacionadas com a ANPC, que vão desde recursos até toda a gestão do teatro de operações. Visto que o sistema foi concebido para dar resposta a ocorrências, grande parte das funcionalidades de simulação acima referidas – simulações e visualização de sistemas CICLOPE – só se encontram disponíveis no contexto de uma ocorrência. Os resultados das simulações são associados a uma ocorrência e podem ser consultados mais tarde. Este é um paradigma que irá ser também reestruturado com a adaptação do sistema às forças de segurança. É importante que as funcionalidades nucleares do sistema estejam disponíveis em qualquer momento e que, para as executar, não se esteja dependente de lógica específica da ANPC (como é o caso da ocorrência).

2.3 Forças de segurança

A origem das forças de segurança em Portugal remonta a um período anterior ao do surgimento da proteção civil. Historicamente, em Portugal a primeira unidade policial terá sido formada pelos quadrilheiros [24], [25] – desde o final da idade média até ao séc. XIX – um corpo de moradores que juravam prestar serviço de policiamento, eram selecionados pela Câmara e serviam durante um período de três anos. O número de quadrilheiros variava de área para área consoante o número de habitantes, e estes eram responsáveis por averiguar crimes e até efetuar prisões de criminosos [26]. Com o aumento populacional o método dos quadrilheiros começava a não ser eficaz para dar resposta com segurança a um número crescente de problemas e, no séc. XIX, os quadrilheiros terão sido extintos, tendo sido formada a polícia cívica [24]. Gradualmente, a polícia cívica divergiu para dar origem às entidades de forças de segurança atuais, sendo que a polícia cívica em si mesma acabaria por dar origem à Polícia de Segurança Pública (PSP), cuja base organizacional é semelhante à da polícia cívica.

De um modo simplista, o papel das forças de segurança é assegurar a segurança interna do país e a ordem pública, através da execução da lei. A execução da lei pode ser garantida através de dois comportamentos possíveis: através de um comportamento de observância das leis pela qual a sociedade se rege, ou através da punição das suas das infrações às mesmas leis [27]. Isto leva-nos aos dois vetores que caracterizam a ação das forças de segurança:

vigilância e atuação sobre os indivíduos. Desde a constituição de 1976, as forças de segurança deixam de ter apenas este papel clássico, passando a ser entidades prestadoras de serviços e auxílio a pessoas, bens de risco ou outras entidades [28]. Legalmente, como estipulado no artigo 25º da Lei de Segurança Interna, "As forças e os serviços de segurança são organismos públicos, estão exclusivamente ao serviço do povo português, são rigorosamente apartidários e concorrem para garantir a segurança interna" [29]. A mesma lei estipula ainda as entidades que são consideradas forças ou serviços de segurança: Guarda Nacional Republicana, Polícia de Segurança Pública, Polícia Judiciária, Serviço de Estrangeiros e Fronteiras e Serviço de Informações de Segurança, assim como órgãos da Autoridade Marítima Nacional e da Autoridade Aeronáutica.

Hierarquicamente, estas entidades estão sob a coordenação do Gabinete Coordenador de Segurança que, por sua vez, respondem diretamente o Primeiro-Ministro ou, por sua delegação, o Ministro da Administração Interna [30], [31]. Em conjunção com a ANPC, algumas destas FS integram a Plataforma Nacional para a Redução de Catástrofes [32], demonstrando os esforços unidos de várias entidades para o combate às emergências.

Faz sentido então a expansão do sistema 4 Forces de forma a englobar as forças de segurança.

Em reunião com um representante das FS, foi apurada a necessidade de um sistema que mantenha as funcionalidades enumeradas – simulações, previsões, videovigilância, etc. – mas que não se restrinja a isso: que seja um sistema que possibilite também a gestão do dia-a-dia das FS. Para dar resposta a este problema surge então o projeto SAFER, que visa expandir o 4 Forces de forma a suportar as necessidades das FS. Gestão de utilizadores, agentes, recursos, ocorrências, missões.

Finalmente, a empresa Action Modulers desenvolveu ainda um *plugin* para gestão de emergências virtuais, *plugin* este que será integrado na aplicação 4 Forces simultaneamente com a integração do projeto SAFER (que consiste na gestão das FS). O *plugin* compõe um módulo de treino para agentes em que é possível avaliar o nível de preparação individual para dar resposta a possíveis emergências.

O modo de funcionamento é simples: é criado um exercício (ou cenário de emergência), composto por situações (que podem ser consideradas ocorrências) que o agente deve ser capaz de resolver corretamente (missão – ou conjunto de tarefas de resposta). O exercício é então executado, havendo duas vistas diferentes: uma para o agente a ser avaliado e outra para o examinador. Durante o exercício, o examinador pode adicionar ou remover em tempo real novas situações, de forma a avaliar a capacidade do agente em se adaptar às mudanças súbitas de cenário no mundo real.

Este *plugin*, apesar de não ter influência direta no projeto SAFER, será uma mais-valia para as FS, permitindo treino regular, dentro do próprio escritório, de forma a manter sempre os agentes o melhor preparados possível para situações de emergência.

Atualmente as FS possuem sistemas dispersos e não integrados. Num contexto exploratório para este trabalho, foram realizados pedidos de contacto com responsáveis da GNR e da PSP em Leiria para averiguar acerca de sistemas em uso atualmente por parte destas entidades. No entanto, devido às normas apertadas que regulam esta informação, não publicamente disponível, não foi possível obter resposta em tempo útil aos pedidos. Anteriormente a este trabalho, foi realizado um estudo acerca das ferramentas utilizadas no Instituto Superior de Ciências Policiais e Segurança Interna, responsável pela formação de agentes policiais. Este estudo revelou que apesar de esta entidade possuir sistemas que permitem a gestão de recursos e equipamentos, a componente operacional – ocorrências e missões – ainda é gerida na sua grande maioria com recurso a ficheiros *Excel* com suporte manual, não havendo instalada uma ferramenta que automatize estes processos (observação: esta informação foi fornecida pessoalmente pelo orientador de estágio na Tecmic).

2.4 Outros sistemas

Esta secção aborda sistemas existentes a nível nacional e internacional que estejam relacionados, seja com a gestão de emergências, seja com a gestão policial.

A área de gestão de emergências é uma área bastante abrangente que engloba desde sistemas mais simples, como gestão de planos de emergência e contactos/notificações, até sistemas mais elaborados que gerem todo o processo de uma ocorrência e ainda alguns que permitem criar simulações de emergências para treino e prevenção. Apesar do projeto SAFER ser orientado a gestão de forças de segurança, este projeto será integrado no *software* XTraN 4 Forces, cujo objetivo fundamental é a gestão de ocorrências. Importa, por isso, estudar o estado da arte como um todo para que se possa avaliar como outras empresas se apresentam no mercado com soluções concorrentes.

Na área da gestão de emergências e proteção civil é identificável um projeto que constitui uma forte entrada no mercado: o projeto SADO - Sistema de Apoio à Decisão Operacional [33], [34]. Este sistema foi desenvolvido pela empresa espanhola Indra [35] para dar suporte à ANPC, substituindo o atual sistema utilizado pela ANPC, o PCGO – Proteção Civil Gestão de Ocorrências. Este sistema visa melhorar a resposta face a situações de risco, através de uma integração entre diversos sistemas dispersos, sendo possível a partilha de informação entre os

diversos agentes da ANPC. Este projeto encontra-se implementado sobre uma arquitetura cliente-servidor, utilizando a plataforma ASP.NET MVC 2.0 para o desenvolvimento, assim como Windows Communication Foundation (WCF) para construir um sistema que possa ser escalável do ponto de vista de comunicação. Para além do SADO, a Indra conta com um historial de projetos no âmbito quer da proteção civil, quer das forças de segurança e defesa interna. Um caso de estudo interessante é o Centro Integrado de Segurança e Emergências de Madrid (CISEM) [36]. O CISEM é um sistema desenvolvido para o ajuntamento de Madrid que permite coordenar esforços de diferentes entidades de modo a reduzir os tempos de resposta perante catástrofes e/ou eventos de grande dimensão. Coordenando entidades como Forças de Segurança e Emergências, Polícia Municipal, bombeiros, etc., consegue-se um sentimento de segurança civil abrangido por mais de quatro milhões de habitantes. A Indra dispõe ainda de um sistema denominado iSafety [37], destinado a centros de comando, contendo capacidade de gestão de crises através de alocação de recursos, gestão de comando, gestão de telefonemas e incidentes, visualização geográfica, etc.

A empresa Ifthen Lda. [38] é uma empresa portuguesa que contém projetos relacionados com bombeiros e proteção civil. Para os bombeiros a Ifthen disponibiliza o sistema IFFIRE [39], para gestão de corporações de bombeiros e, segundo o *site* da empresa, contando com mais de 80% de quota de mercado em Portugal. Uma vez que os bombeiros operam em última análise sob a ANPC, o IFFIRE é uma solução interessante para ser descrita neste relatório. O IFFIRE permite gerir toda a lógica necessária de corporações de bombeiros, desde bombeiros, viaturas e incidentes até operações de gestão e faturação. Este sistema é tipicamente apresentado como um conjunto de três ou mais máquinas ligadas em rede – comando, área operacional, área administrativa, outros – dependendo da organização da corporação. O sistema assenta em tecnologias Microsoft, sendo o sistema operativo Windows (versão 95 ou superior) um pré-requisito para o sistema.

Também da empresa Ifthen existe o projeto IFPROTEC [40], vocacionado, segundo palavras do *site*, "para as Proteções Civis Portuguesas e outras entidades que intervêm na prevenção e coordenação de meios e recursos". O IFPROTEC permite a gestão de meios humanos, veículos, entidades de intervenção, planos de emergência, etc. Este sistema permite o acompanhamento de todo o processo de ocorrências, permitindo ainda realizar gestão documental sobre os processos. À semelhança do IFFIRE, o IFPROTEC funciona em ambiente Windows e utiliza bases de dados SQL Server para persistência de dados ou Microsoft SQL Server Data Engine (MSDE).

Relacionados com forças policiais ou forças de segurança em geral existem vários tipos de

projetos, desde projetos orientados à gestão de cadastros criminais, portais de divulgação de crime para o público, projetos de gestão de ocorrências (RMS – Records Management System), entre outros. O projeto Gesnpolicia [41] é um projeto destinado às forças policiais e autarquias para gerir os números de polícia – identificação de casas num espaço público. Este projeto permite que as autarquias possam gerir as suas toponímias, sendo uma fonte interessante de dados para outros sistemas. Este sistema funciona no formato de *website*.

Um projeto interessante datando de 2005, é o Blue CRUSH [42], que foi implementado pela polícia de Memphis em parceria com a universidade da Califórnia para a redução de crimes, essencialmente dos crimes mais violentos [43]. Utilizando um *software* da IBM para análise de dados e predição de resultados, foi possível reduzir os crimes através da análise de histórico de ocorrências. Esta análise permitiu a realocação de recursos de um modo mais pertinente e eficaz.

O papel das Forças de Segurança não se restringe a tratar de ocorrências, havendo também um papel preventivo na tentativa de reduzir o número das mesmas, principalmente as mais graves, que são originadoras dos maiores danos à sociedade. Daí a existência de projetos como o CrimeMapping [44], CrimeReports [45] ou ainda PublicEye [46]. Os projetos CrimeMapping e CrimeReports são *websites* que têm uma finalidade muito semelhante: integrar informação sobre ocorrências de caráter criminoso e disponibilizar essa informação para o público de modo a criar maior consciência acerca dos perigos existentes para se poder reforçar a segurança. A título de exemplo, consideremos o seguinte caso: um determinado parque tem um elevado número de registos de crime sexual. Esta informação tem duas interpretações diferentes que podem levar à diminuição destes crimes: a visão das Forças de Segurança e a visão do público. Para as FS é sinal que a área não está a ser vigiada devidamente sendo, por isso, necessário uma vigilância mais apertada. Para os cidadãos, permite que estes tenham conhecimento desse facto. Seja qual for o ponto de vista, o facto de a informação estar visível para todos pode ajudar a prevenir crimes, sendo estes sistemas uma mais-valia para a população. O projeto PublicEye baseia-se no mesmo princípio, mas estende o conceito. Em vez de se limitar à disponibilização desse tipo de informação aos cidadãos, permite que estes participem. Este projeto, disponível sob a forma de aplicações para as plataformas móveis Android, iOS e Windows Phone, visa tornar a segurança pública quase numa rede social. Apesar de qualquer um poder contribuir, existem perfis de utilizadores – polícia, bombeiros, etc. – que tratam da grande maioria da informação, através da integração com sistemas Computer Aided Dispatch (CAD) e RMS existentes, o que resulta em grande parte da informação ser disponibilizada de modo automático, mas permitindo que utilizadores

“comuns” possam contribuir, com fotos, vídeos, etc. sobre novas ocorrências. Estas fotos/vídeos podem servir como *input*, a agentes que se dirijam para o local da ocorrência, para que estes cheguem ao local com mais informação sobre a ocorrência. O sistema conta ainda com um conjunto promissor de ferramentas como *streaming* de vídeo, notificações avançadas, variados tipos de comunicação, navegação no mapa, etc.

Contudo, a prevenção não basta, e é necessário gerir ocorrências de uma forma sustentável, e neste contexto existe *software* RMS, destinado (não exclusivamente) à gestão de ocorrências. São vários os exemplos de RMS, entre os quais o CrimeStar [47], o Spillman Police Records Software [48], o ledsSuite [49], o Phoenix Police Records Management System [50], entre outros. O CrimeStar é um sistema modular, com suporte a gestão de ocorrências e casos policiais, gestão prisional, gestão documental, mandatos, *profilling*, etc.

O sistema Police Records Software da empresa Spillman encontra-se equipado com um leque de funcionalidades bastante completo, desde a gestão de ocorrências até toda a gestão da evolução dos casos. Este sistema conta com funcionalidades de gestão de comando, equipamentos, veículos, visualização de informação de trânsito, gestão de provas, gestão de patrulha, visualização de informação em sistemas geográficos em tempo real, capacidade de pesquisa avançada em várias fontes de dados (entidades externas), deteção de informação duplicada, gestão de processos com recurso a *workflows*, etc. Este sistema suporta adição de módulos, consoante as necessidades.

O sistema ledsSuite é um sistema composto por módulos separados que formam uma aplicação bastante completa. Os módulos deste sistema incluem RMS para gestão de ocorrências, CAD para auxílio de alocação de recursos no terreno e gestão de unidades no terreno, visualização com recurso a GIS (Geographic Information System), gestão prisional, gestão documental, etc. Este sistema contém ainda um módulo administrativo e uma forte componente de relatórios que integra com os vários módulos do sistema.

O *software* Phoenix Police Records Management System é um sistema que, para além da capacidade de gerir ocorrências, agentes, veículos, evolução de casos policias, gestão documental, etc., contém a vantagem de estar também disponível para dispositivos móveis do tipo PDA, para utilização no terreno pelos agentes. Este sistema conta ainda com uma componente de Business Intelligence bastante forte. O sistema principal é baseado nas tecnologias .NET que e funciona através de um *web browser*. Para uma análise de outros sistemas consultar a referência [51]. O próximo capítulo descreve a metodologia concebida para o desenvolvimento deste projeto, referindo ainda o planeamento realizado na área de gestão do projeto.

3 Metodologia e Gestão de Projeto

Este capítulo descreve a metodologia concebida para utilizar no desenvolvimento do projeto SAFER, assim como o planeamento de projeto realizado.

3.1 Metodologia

Com o objetivo de auxiliar o desenvolvimento do projeto SAFER, foi necessária a definição de uma metodologia de suporte ao desenvolvimento do projeto. A metodologia concebida é baseada no modelo de desenvolvimento em cascata [52], que define que etapas relativas ao desenvolvimento de *software* ocorrem de um modo sequencial e não reversível. Tal como a água que cai da cascata, que não volta a subir, uma vez dado início a uma determinada fase de desenvolvimento, não é possível voltar à fase anterior segundo o modelo. Este modelo é aplicável quando são conhecidos à partida todos os *inputs* e *outputs* e aspetos a integrar no sistema [53]. Sendo impossível saber à partida todos os problemas que irão ocorrer durante o processo de desenvolvimento, o modelo em cascata é frequentemente tido como um modelo irrealista, devido ao facto de ser puramente unidirecional [54]. Este modelo constitui, todavia, o ponto de partida que levou ao surgimento de várias outras metodologias [55], [56]. Também a metodologia concebida para suportar o desenvolvimento do SAFER é baseada neste modelo, contendo algumas adaptações, que permitem a navegação bidirecional entre etapas de desenvolvimento. Esta navegação bidirecional é justificável pelo contexto empresarial em que se insere o projeto, ambiente em que as alterações de requisitos mudam com alguma frequência, de forma a adaptar-se às várias necessidades dos clientes. Nenhuma das etapas é imune a erros, e a navegação entre etapas de forma unidirecional implicaria acarretar esses mesmos erros para etapas futuras, o que não é desejável, principalmente quando os erros ocorrem numa fase inicial do projeto. Um exemplo típico refere o processo de análise, cujos erros nesta etapa tipicamente só são descobertos na fase de implementação, invalidando muitas vezes parte do trabalho, sendo necessário repensar o problema para balançar o esforço

com o objetivo em causa.

Tipicamente, a navegação entre etapas é realizada entre etapas sucessivas e no sentido descendente, como se de uma cascata se tratasse. Em cada fase é refinada a etapa até obter um resultado desejável, procedendo-se para a etapa seguinte.

É comum, ao mudar de etapa, chegar-se à conclusão de que a etapa anterior apesar de trabalhada e refinada, ainda necessita de mais afinações. A metodologia apresentada permite voltar a uma etapa anterior, de forma a não se transportar um erro de uma etapa para a etapa seguinte.

Por vezes, é necessário repensar o problema, sendo para isso necessário voltar atrás várias etapas. É preferível recuar e perder algum tempo a repensar o desenvolvimento futuro do que implementar sobre uma base que contenha falhas que mais tarde serão um impedimento com um custo maior para o projeto. Note-se que nesta metodologia, apesar de ser possível retroceder mais do que uma etapa de uma só vez, não é possível avançar para uma etapa futura sem passar, por ordem, pelas etapas anteriores. A Figura 2 ilustra a metodologia utilizada para a elaboração do projeto:

- Prototipagem: antes da implementação das componentes, é realizada uma prototipagem dos mesmos. Os protótipos referidos por esta etapa não se restringem a desenhos visuais, mas também a pequenas aplicações de teste exploratórias como forma de abordar os conceitos novos a aplicar. Neste projeto foram desenvolvidas aplicações de teste, realçando-se uma primeira para exploração da criação de *webservices* (para a componente *web*), e uma segunda que consistiu na criação de um *plugin* para interiorização do sistema de *plugins* utilizado no 4 Forces. Estas aplicações são determinantes para o desenvolvimento do sistema, atuando como testes antes da implementação, servindo ainda para futura consulta. Como não poderia deixar de ser, é nesta etapa que são também desenhados protótipos não funcionais, que servem como linhas orientadoras aquando da implementação dos formulários do *plugin*.
- Implementação: esta é a fase em que é escrito o código das aplicações que fazem parte do sistema final. O código escrito está todo em inglês, segue as normas de escrita para as linguagens de programação em utilização e o código tem sempre comentários descritivos, num formato de sumário/*javadoc* para todas as classes, propriedades e métodos. Este processo garante que o código está sempre documentado. É também na fase de implementação que é realmente validado o resultado da etapa da prototipagem.

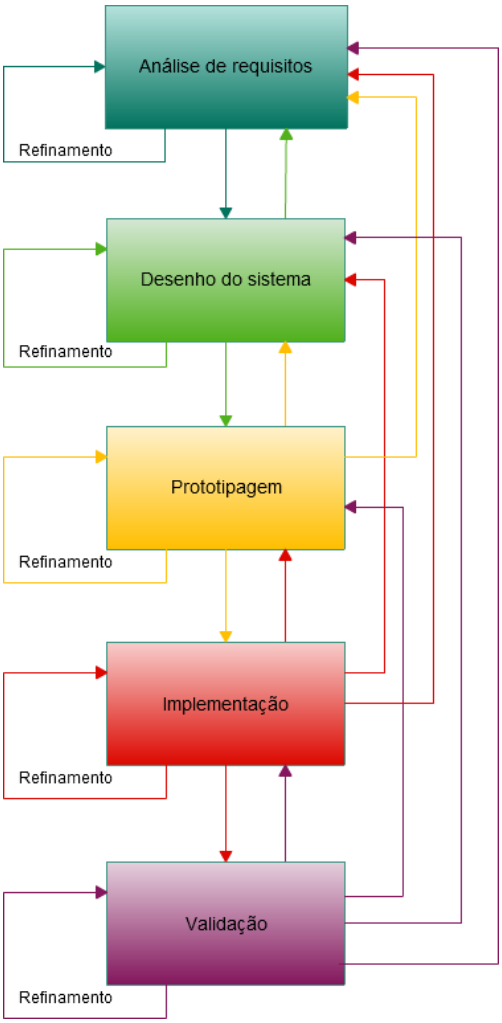


Figura 2 - Metodologia de desenvolvimento utilizada.

Frequentemente surgem problemas que requerem a criação de novas aplicações de teste de forma a validar novas bibliotecas ou APIs (*Application Programming Interfaces*) antes de as inserir no sistema. Esta etapa está fortemente ligada à etapa anterior e à etapa que se sucede, tendo ocorrido várias vezes navegação bidirecional entre as mesmas. Os resultados desta etapa são validados através da etapa de validação.

- Validação: esta etapa consiste maioritariamente em testes de aceitação internos, realizados aquando da implementação de novas funcionalidades para validar o estado do desenvolvimento. Em adição é realizada bissemanalmente uma depuração, na qual é testado o sistema com maior profundidade, transversalmente às várias componentes desenvolvidas. Esta depuração é composta pela realização de testes com o objetivo de averiguar o comportamento do sistema perante situações menos vulgares. Neste processo é encontrada a maioria dos problemas, que são então apontados para futura resolução. Uma vez concluída, esta etapa pode implicar um retorno à etapa anterior

com o objetivo de resolver os problemas encontrados e desenvolver novas funcionalidades.

O desenvolvimento mantém-se até à obtenção de resultados estáveis e satisfatórios que podem de forma segura ser integrados no sistema 4 Forces.

3.2 Gestão de Projeto

O estágio teve início a 3 de Setembro de 2012 e terminou com a entrega deste documento, a 31 de Outubro de 2013. Tendo o estágio decorrido num contexto empresarial, foi necessária a existência de um período de adaptação, com a finalidade de integração na equipa de desenvolvimento, nas principais instalações da empresa, no Taguspark (Oeiras).

As primeiras semanas consistiram na instalação de *software* de forma a criar uma base de trabalho sólida, com todas as ferramentas de trabalho necessárias, assim como na formação interna acerca dos vários produtos, já implementados ou ainda em desenvolvimento, que constituem o repertório da Tecmic. Este período permitiu obter uma melhor perceção acerca do modelo de negócio da empresa.

Além disso, também como forma de facilitar a adaptação à empresa, foi proposta a integração num projeto a decorrer na altura, o projeto da Empresa da Eletricidade da Madeira (EEM), cuja finalidade consistiu em desenvolver um sistema de gestão de equipas. Este sistema destina-se a ser utilizado por técnicos no terreno na recolha de leituras, averiguação de incidentes e reparação de material elétrico.

A contribuição no estágio para este projeto incidiu na aplicação para terminais móveis, utilizada pelos agentes em serviço. Sendo um projeto complexo, o trabalho neste projeto prolongou-se até ao final de janeiro de 2013. A formação acerca do estado do 4 Forces teve início ainda nesse mesmo mês e o início dos trabalhos no SAFER teve início em fevereiro, altura em que o local de trabalho passou para Leiria, onde foi realizado o resto do estágio, assim como todo o trabalho relativo ao SAFER.

Para uma análise das tarefas realizadas neste estágio, consultar o Anexo II que contém uma lista das tarefas relativas ao período passado na Tecmic, organizada devidamente com base no tipo de trabalho cumprido. Sobre o SAFER, encontra-se disponível no Anexo III um gráfico de Gantt.

O próximo capítulo descreve em detalhe o desenvolvimento do projeto SAFER, justificando as opções tomadas durante o mesmo.

4 Desenvolvimento

Este capítulo descreve o desenvolvimento do sistema ao longo do estágio, no âmbito do projeto SAFER. Começa por introduzir as tecnologias e ferramentas utilizadas nas várias componentes desenvolvidas, seguindo-se a estruturação da solução do 4 Forces, apresentação da arquitetura geral, procedendo-se à descrição do desenvolvimento propriamente dito das componentes do SAFER, decisões tomadas e implementação.

4.1 Tecnologias utilizadas

O desenvolvimento do SAFER envolve várias tecnologias e ferramentas por ser um sistema composto por várias componentes distintas.

O 4 Forces é um sistema maioritariamente desenvolvido utilizando a plataforma (*framework*) .NET [57], ambiente de desenvolvimento de eleição utilizado na Tecmic. De modo a facilitar a integração entre sistemas e a manter o mais possível o projeto conforme com as normas Tecmic, o projeto SAFER é também desenvolvido sobre a mesma plataforma.

O desenvolvimento é realizado com recurso ao *Integrated Development Environment* (IDE) Visual Studio [58], sistema proprietário da Microsoft [59] para realização de desenvolvimento. O 4 Forces conta com projetos (daqui em diante o termo projeto refere um projeto criado no Visual Studio, e o termo solução refere o conjunto de projetos que juntos formam o 4 Forces, também dentro do Visual Studio) criados ao longo de vários anos e em diferentes versões da *framework* .NET, sendo a mais antiga a versão 3.5 e a mais recente a versão 4.0. A versão 4.0 é utilizada na quase totalidade da solução O restante corresponde a projetos que, por motivos de retrocompatibilidade, não podem ser atualizados por serem bibliotecas transversais utilizadas por vários produtos da Tecmic, sendo o custo da mudança considerável.

Aquando do início do estágio, a versão do Visual Studio utilizada no projeto 4 Forces era a de

2010. No início dos trabalhos foi realizada, sem grande esforço, a migração para a versão 2012. Esta migração justifica-se por vários motivos, entre os quais a compatibilidade da solução entre ambas as versões, salvo raras exceções; a possibilidade de utilização de um IDE mais recente e com novas ferramentas e ainda a capacidade de criação de novos tipos de projeto (útil para desenvolver a componente servidor, como será abordado mais à frente).

O 4 Forces é um sistema complexo e de grande dimensão, tendo sido desenvolvido ao longo do tempo por várias pessoas – e até por várias entidades externas à Tecmic, como a Action Modulers e o INOV. Tendo sido desenvolvidos por diferentes pessoas com diferentes gostos pessoais, as linguagens de programação utilizadas não são as mesmas em todos os projetos.

Maioritariamente, o desenvolvimento é realizado com recurso à linguagem de programação C#, existindo todavia alguns projetos desenvolvidos utilizando Visual Basic .NET. Uma vez que todo o código escrito em qualquer linguagem da Framework .NET é convertido para Common Language Runtime (CLR) – a linguagem intermediária executada pela Virtual Machine da plataforma .NET – é garantida a compatibilidade entre projetos apesar das diferentes linguagens de programação utilizadas. Para o desenvolvimento do SAFER a linguagem escolhida foi C# visto ser a linguagem mais utilizada no 4 Forces.

4.2 Arquitetura do 4 Forces

Esta secção serve o propósito de descrever o estado do 4 Forces antes do projeto SAFER, assim como a sua arquitetura geral de alto nível.

O sistema 4 Forces encontra-se desenvolvido como uma solução do Visual Studio única contando, antes do desenvolvimento do SAFER, com cerca de 120 projetos diferentes. O tipo e complexidade dos projetos variam bastante. Apenas como ilustração, existem projetos com menos de cinco classes e projetos com mais de trinta.

Por tipo de projeto, a solução pode ser dividida em secções como modelos de domínio variados (com a respetiva camada de acesso a dados e lógica de negócio), bibliotecas de controlos reutilizáveis (utilizadas na construção de *forms*), bibliotecas com *forms* variados não diretamente reutilizáveis por serem específicos, *plugins* a integrar na aplicação principal, bibliotecas de execução de simulações, acesso a videovigilância (e outros dados relevantes), serviços para obtenção de dados, bibliotecas partilhadas com outras soluções Tecmic, bibliotecas de integração, projetos de testes e bibliotecas de utilitários. A Figura 3 demonstra, de uma forma bastante simplificada, a categorização desses mesmos projetos segundo o seu tipo. Esta categorização não é de forma alguma exaustiva, existindo projetos que se

enquadram facilmente em mais do que uma categoria. Esta categorização é apresentada de forma a dar a conhecer um pouco acerca da estrutura da solução.

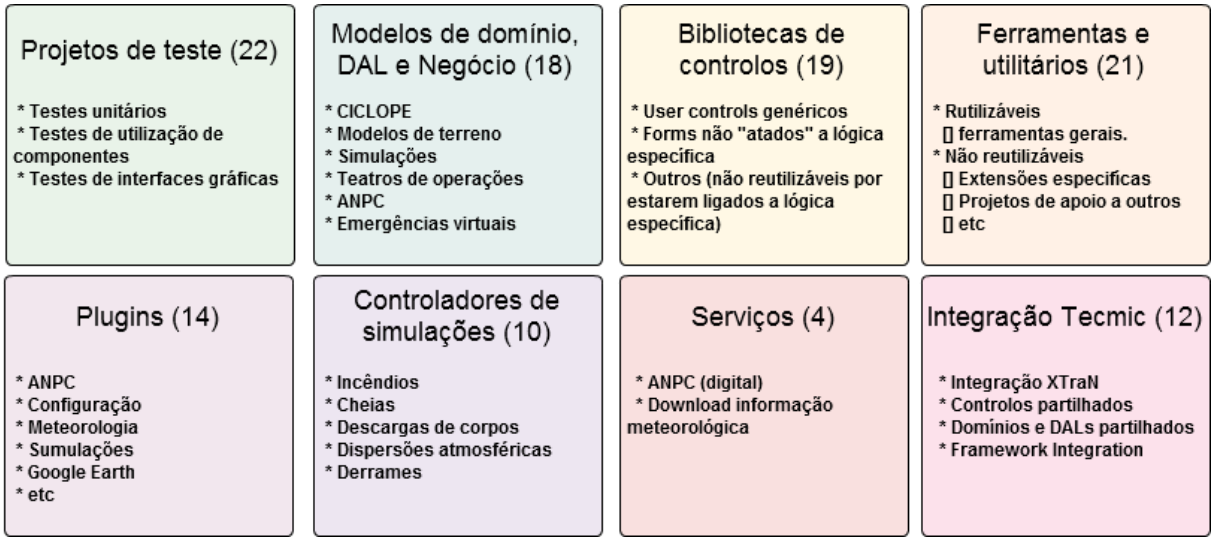


Figura 3 - Categorização dos projetos por tipo.

Arquitecturalmente, o 4 Forces está construído em volta da aplicação principal (designada de *Main Application*). A esta aplicação são adicionados *plugins*, e é nestes *plugins* que é adicionado o valor à aplicação. A figura seguinte apresenta o diagrama arquitetural de alto nível do sistema:

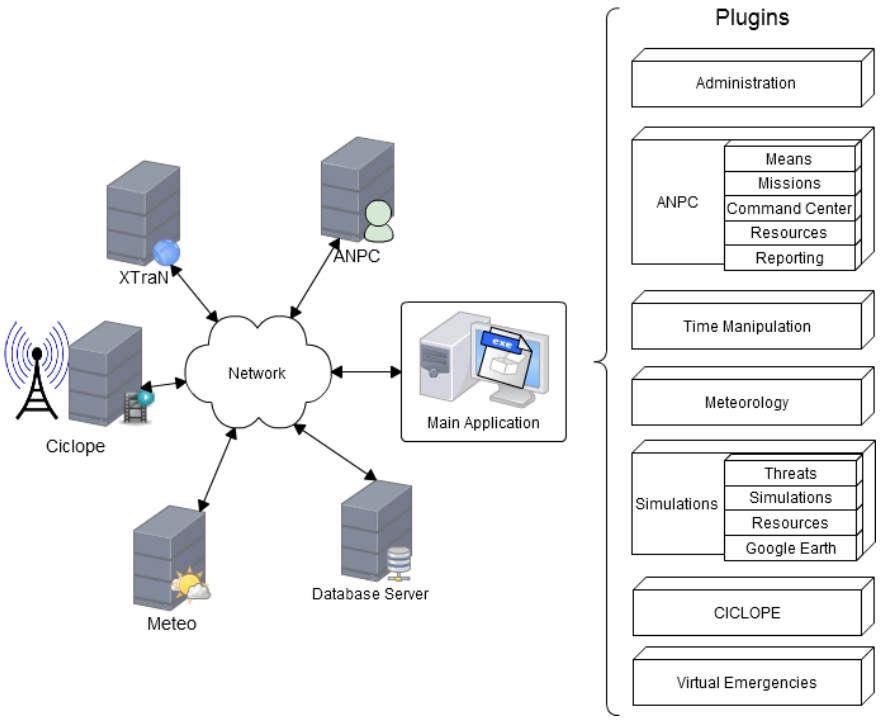


Figura 4 - Arquitetura de alto nível XTraN 4 Forces.

A figura encontra-se dividida, contendo no lado esquerdo as fontes de dados externas (incluindo servidor de bases de dados) sobre as quais o sistema atua e, no lado direito, uma

representação dos *plugins* existentes no 4 Forces.

A *Main Application*, apesar de estar representada como ligada às fontes externas, não realiza contacto direto com as mesmas, sendo a comunicação realizada através dos *plugins*. Cada *plugin* tem conhecimento acerca das suas próprias fontes e/ou persistência de dados, e a comunicação com as mesmas fontes é transparente para a *Main Application*. Aliás, a própria aplicação não sabe da existência de nenhum *plugin* em específico, apenas possui lógica que permite o carregamento dos mesmos.

Os *plugins* (ou módulos) encontram-se na mesma figura divididos consoante a sua funcionalidade: administração, ANPC, manipulação temporal, meteorologia, Simulações/Modelações, CICLOPE e emergências virtuais.

O módulo administrativo permite configurar dados cartográficos, assim como outras configurações internas, transversais a outros *plugins.* Na secção da ANPC existem vários submódulos, responsáveis por gerir meios, recursos, centros de comando, missões, e ainda criar relatórios orientados à ANPC. O módulo de controlo temporal permite gerir o intervalo de tempo sobre o qual o sistema está a trabalhar. Isoladamente, a sua utilidade é bastante reduzida. No entanto, torna-se uma das peças chave do sistema em conjunto com os módulos de meteorologia e de simulações. No caso da meteorologia, permite controlar os períodos para os quais a informação meteorológica é apresentada e no caso das simulações permite visualizar os resultados das simulações ao longo do tempo, de forma a observar a evolução das simulações. O módulo meteorológico permite obter e apresentar dados meteorológicos, tais como velocidade e direção do vento, pluviosidade, correntes marítimas, temperaturas, etc. Se este *plugin* não estiver carregado, não é possível "alimentar" as simulações com dados fidedignos, podendo no entanto ser utilizados dados fictícios para simular uma dada situação.

O módulo de simulações é um conjunto de *plugins* que permite a definição de ameaças (incêndio, cheia, dispersão atmosférica, etc.), configurar os dados de origem, correr e visualizar os dados. Permite ainda exportar os resultados e visualizar os mesmos no Google Earth (inclusivamente, o Google Earth está ainda incorporado no próprio *plugin*), em adição ao mapa 2D. Segue-se o módulo do CICLOPE, que permite comunicação com as torres de videovigilância florestal e visualização dos dados das mesmas. Por último, mas não menos importante, o módulo de emergências virtuais serve o propósito de treino e avaliação de agentes dentro do escritório para situações que possam vir a acontecer, tal como explicado no capítulo 2.

Idealmente, salvo raras exceções, os *plugins* deveriam ser independentes e funcionar isoladamente. Contudo, por motivos históricos e devido à elevada rotatividade de pessoal

dentro do projeto e urgência na apresentação de um produto funcional, o sistema tornou-se aos poucos praticamente todo orientado à ANPC, existindo dependências em quase todos os módulos, de lógica e modelo de domínio específicos da ANPC. Este facto constitui atualmente a maior limitação ao desenvolvimento do SAFER.

4.3 Modelo de Domínio

Após a análise de requisitos foi concebido um modelo de domínio capaz de suportar a lógica das Forças de Segurança. No Anexo I é possível consultar o documento que contém a análise de requisitos realizada.

Numa fase inicial pensou adaptar-se o modelo existente da ANPC e adicionar apenas as entidades novas. No entanto, esta abordagem foi descartada após um estudo de viabilidade. Para além do tempo que seria necessário para compreender em profundidade o modelo da ANPC e quais as implicações da alteração de cada entidade (do modelo) no sistema existente, existe ainda toda uma complexidade de dependências entre projetos (no Visual Studio) que tornariam esta abordagem pouco viável.

Assim, foi desenhado um modelo de domínio capaz de suportar a lógica das FS, tendo sido estruturado de forma a ser o mais genérico possível, para que no futuro se possa estruturar o atual modelo da ANPC para estar conforme com este, sem prejuízo de lógica específica. Este modelo de domínio foi desenvolvido como um conjunto de objetos Plain Old CLR Objects (POCO), designação atribuída geralmente a objetos que podem ser persistidos mas que não se encontram acoplados a nenhuma ferramenta de Object-Relational Mapping (ORM) específica. Sendo um modelo composto apenas por objetos POCO, este modelo não se encontra dependente de *Attributes*, os objetos não necessitam de herdar nenhuma classe em particular ou implementar uma determinada interface imposta por uma ferramenta de ORM. O modelo não tem, assim, qualquer conhecimento de que será utilizado por uma ferramenta de ORM, à exceção de um detalhe: todas as propriedades e métodos das entidades (daqui em diante, os objetos do modelo de domínio serão referidos com entidades) têm de ser declarados como *virtual*, para que subclasses possam efetuar *overriding* dos mesmos (o motivo é explicado detalhadamente mais à frente neste capítulo).

O diagrama geral do modelo está apresentado na Figura 5. As entidades representadas estendem, em última análise, da classe EntityBase. Esta classe disponibiliza uma base para qualquer entidade do sistema. Salientam-se os campos Id, data de última modificação e uma *flag* que indica se o objeto se encontra removido ou não. Assim, com o auxílio da classe

EntityBase, garante-se que todas as entidades seguem estas normas, sendo possível a implementação de comportamentos genéricos nos clientes. A título de exemplo, quando um cliente obtém dados pode comparar os dados anteriores, pelas propriedades Id e LastModificationDate, de modo a decidir se atualiza a interface gráfica ou não.

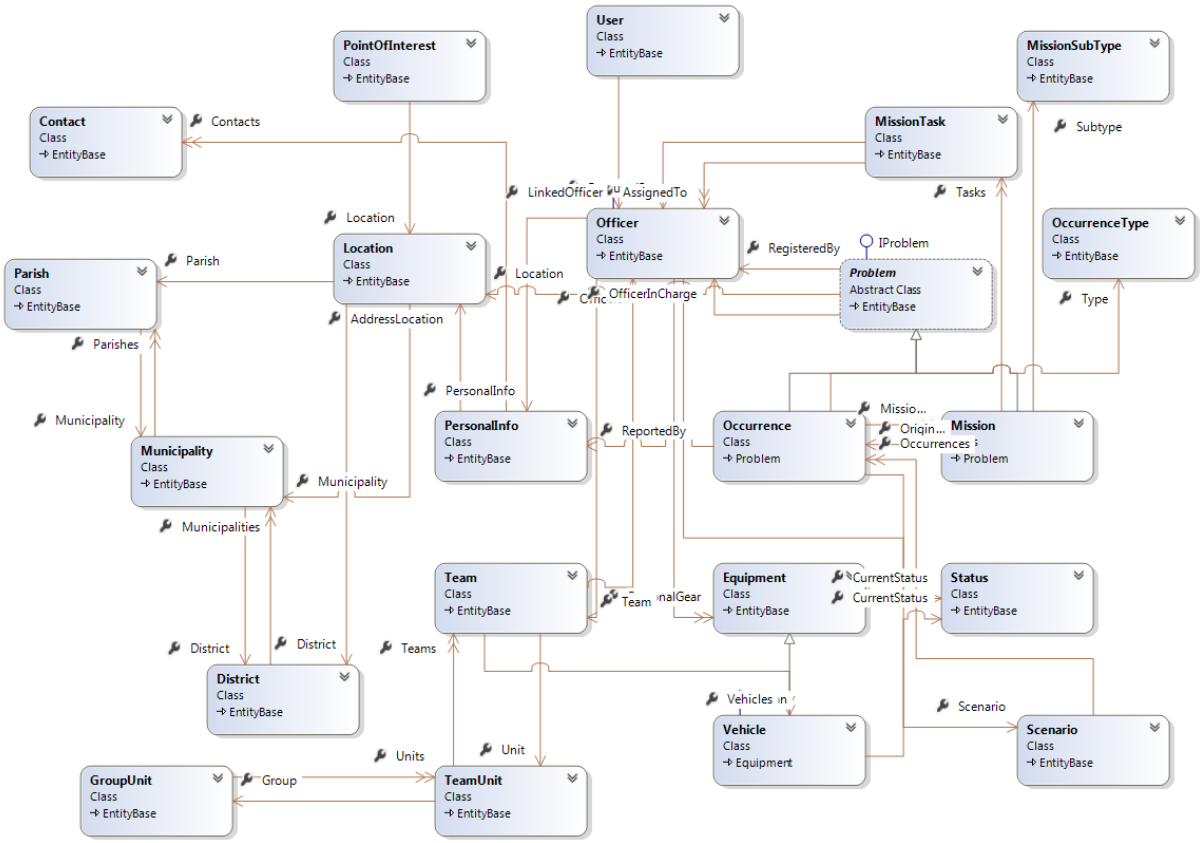


Figura 5 - Diagrama do modelo de domínio (completo).

O elevado número de relações entre as classes do modelo dificulta a compreensão da figura anterior. Por este motivo, este subcapítulo segmenta este diagrama, dando a conhecer detalhadamente o modelo e as decisões associadas. Este subcapítulo não visa dar a conhecer em detalhe todas as propriedades das entidades mas, sim, descrever as propriedades que compõem relações entre as diversas entidades do modelo:

- User: entidade que representa um utilizador no sistema, cujas dependências são ilustradas na figura que se segue:

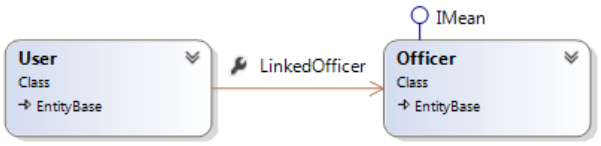


Figura 6 - Entidade User.

A gestão de utilizadores é importante para permitir autenticação no sistema. Um utilizador do sistema pode (ou não) ser um agente conforme representa a propriedade LinkedOfficer no

utilizador.

- Officer: a entidade que representa um agente. As dependências da classe que representa o agente encontram-se de seguida ilustradas:

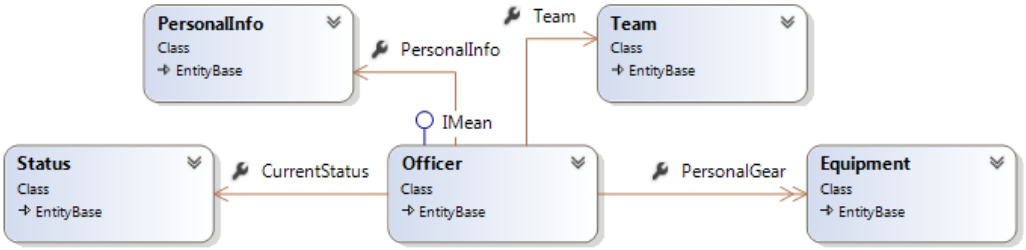


Figura 7 - Entidade Officer.

O agente relaciona-se com várias entidades do modelo. Um agente encontra-se ligado à classe PersonalInfo, onde obtém a informação da pessoa associada ao agente. Um agente pode pertencer a uma equipa e contém uma lista de equipamentos associados ao mesmo. Uma vez que Officer implementa IMean também tem a propriedade CurrentStatus, que representa o último estado conhecido do meio (informação geográfica, data, etc.).

- Status: Esta entidade não depende de nenhuma outra entidade do modelo. Serve para representar o estado atual de um meio, incluindo informação geográfica associada ao mesmo.
- PersonalInfo: entidade que representa a informação pessoal de um indivíduo.

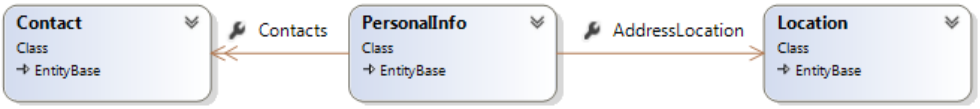


Figura 8 - Entidade PersonalInfo.

Esta classe não representa o conceito de pessoa, mas sim de informação pessoal. Uma pessoa tem uma informação pessoal – é este o conceito desta entidade. Uma informação pessoal tem uma lista de contactos a partir dos quais a pessoa pode ser contactada, assim como uma localização para morada.

- Contact: classe que representa um contacto a partir do qual uma pessoa pode receber ou efetuar comunicações. Esta classe não depende de nenhuma outra do modelo de domínio.
- Location: entidade que representa uma localização no mapa. As dependências desta entidade de caráter geográfico encontram-se ilustradas na figura seguinte:

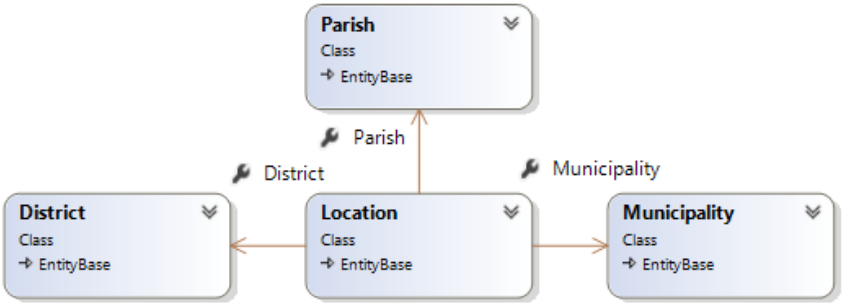


Figura 9 - Entidade Location.

Uma localização tem coordenadas únicas. Para auxiliar o processo de manipulação da informação da localização, esta define a freguesia a que pertence. As restantes dependências – concelho e distrito – são obtidas a partir da freguesia a que o objeto pertence.

- PointOfInterest: esta entidade representa um ponto de interesse geográfico. A nível de dependências, apenas depende da entidade Location. Estes pontos podem ser hospitais, estádios, redes elétricas, pontos turísticos, etc.
- Parish, Municipality, District: Estas entidades representam, respetivamente, uma freguesia, um concelho e um distrito, cuja relação se encontra ilustrada na figura seguinte:

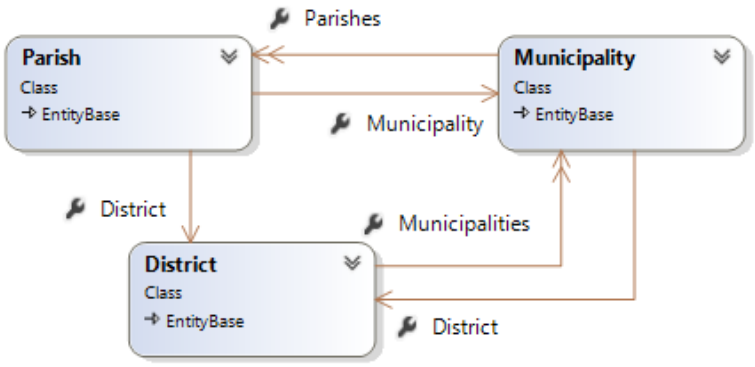


Figura 10 - Entidades Parish, Municipality e District.

No topo da hierarquia encontra-se o distrito, no fundo a freguesia. Um distrito contém uma lista de concelhos que pertencem ao distrito. Um concelho contém o distrito a que pertence, assim como uma lista de freguesias afetas. Por sua vez uma freguesia tem o concelho a que pertence e – indiretamente – um distrito.

- Team: esta entidade representa um agrupamento de meios. As dependências desta entidade encontram-se esquematizadas de seguida:

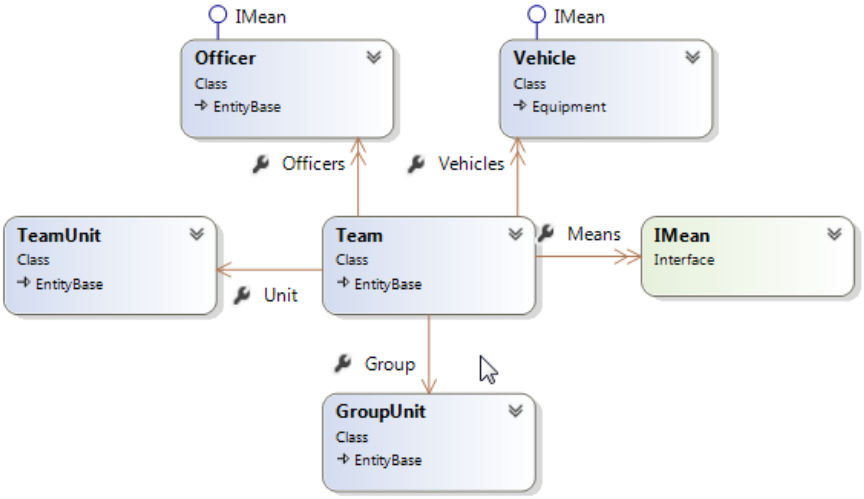


Figura 11 - Entidade Team.

A equipa é um agrupamento de meios. Estes meios podem ser do tipo agente – Officer – ou do tipo Veículo – Vehicle. Uma equipa pode pertencer a uma Unidade (TeamUnit), que por sua vez pode pertencer a um grupo.

- GroupUnit, TeamUnit e Team: à semelhança do triângulo distrito/concelho/freguesia este conjunto de entidades funciona de modo semelhante mas para organizar – em vez de localizações – meios.

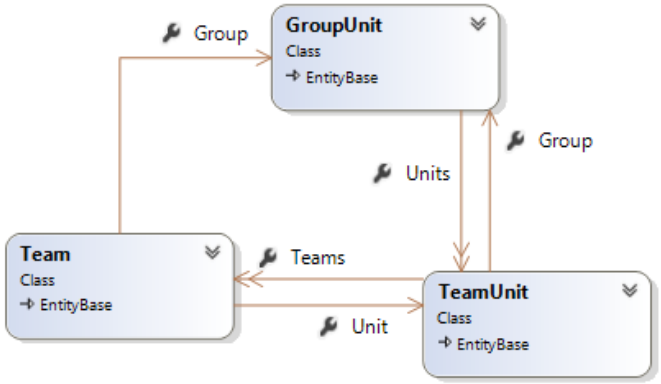


Figura 12 - Entidades GroupUnit, TeamUnit e Team.

Uma equipa pode pertencer a uma unidade e, consequentemente, a um grupo. Uma unidade depende do grupo a que pertence e contém uma lista de equipas afetas. O grupo apenas contém uma lista de unidades que lhe estão afetas, encontrando-se no topo da cadeia.

- Equipment & Vehicle: Estas entidades representam os equipamentos utilizados pelos agentes. A hierarquia encontra-se representada na figura que se segue:

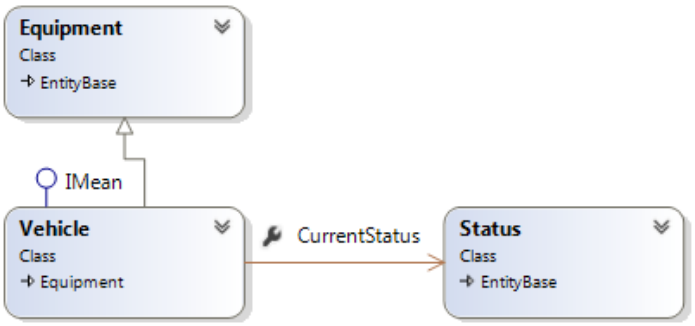


Figura 13 - Entidades Equipmnet e Vehicle.

Um equipamento não se encontra dependente de outras entidades do modelo. A entidade Vehicle é um equipamento que implementa IMean tendo um estado associado.

- Problem: tanto as ocorrências como as missões são consideradas "problemas", tendo um estado, uma localização, e datas e agentes responsáveis. As dependências de problemas encontram-se ilustradas na figura seguinte:

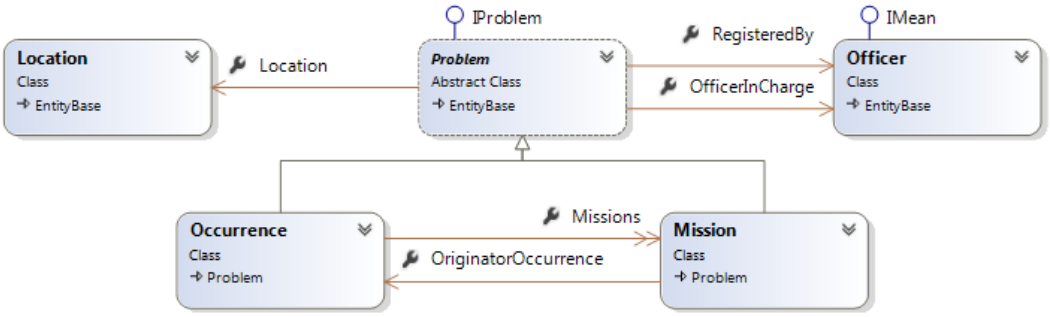


Figura 14 - Entidade Problem.

A classe abstrata Problem (que implementa a interface IProblem) encontra-se na base das ocorrências e das missões. Concetualmente, uma ocorrência é o registo de um acontecimento (v.g. "Assalto à mão armada no banco da Praça Rodrigues Lobo"), enquanto que uma missão é uma ação. Uma missão pode ser do tipo reativa – resposta a uma ocorrência – ou do tipo proativa – prevenção.

- OccurrenceType: esta entidade representa tipos de ocorrências, não apresentando dependências de outras entidades do modelo de domínio. Um tipo de ocorrência define um esqueleto contendo tarefas tipo a executar na missão reativa correspondente. A título de exemplo, para um tipo de ocorrência "ameaça de bomba" o esqueleto de tarefas pode conter "reportar oficial", "chamar brigada anti minas", etc.
- Occurrence: esta entidade representa uma ocorrência – um acontecimento importante o suficiente para merecer registo, bem como informação relevante sobre a mesma. Tal como ilustrado na figura seguinte, tipicamente uma ocorrência pode dar origem a uma ou mais missões:

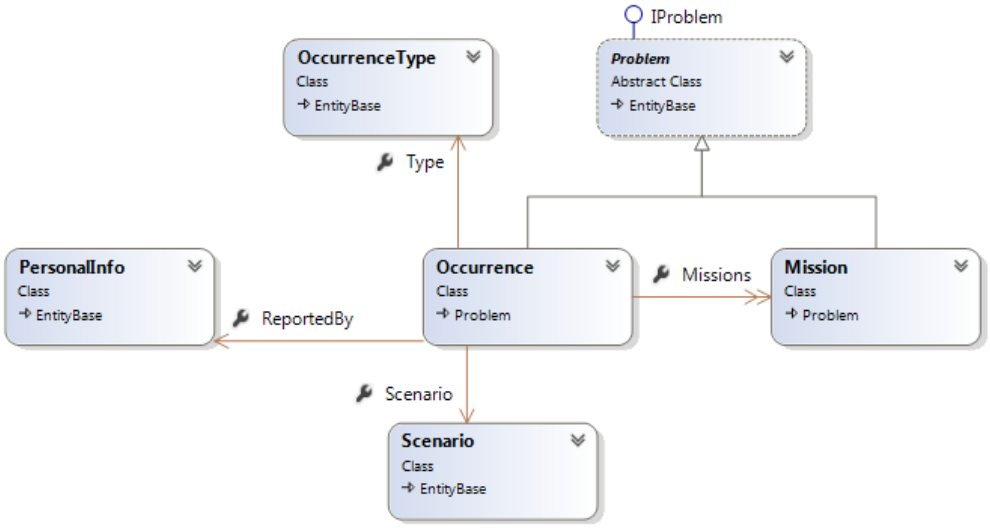


Figura 15 - Entidade Occurrence.

Além de missões, uma ocorrência contém o campo ReportedBy, que consiste na informação da pessoa que reportou a ocorrência. Uma ocorrência pode pertencer ainda a um cenário (agrupamento de ocorrências).

- Scenario: esta entidade representa um agrupamento de ocorrências que se relacionam, tal como é ilustrado na figura que se segue:

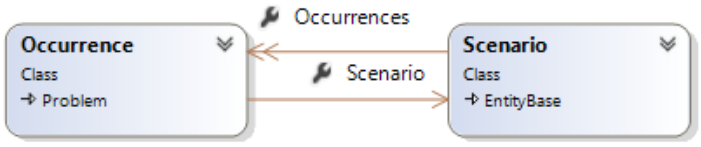


Figura 16 - Entidade Scenario.

Um evento de larga escala pode dar origem a várias ocorrências que, para melhor organização, podem ser agrupadas num cenário.

- Mission: esta entidade representa uma missão: uma ação que é executada por agentes.

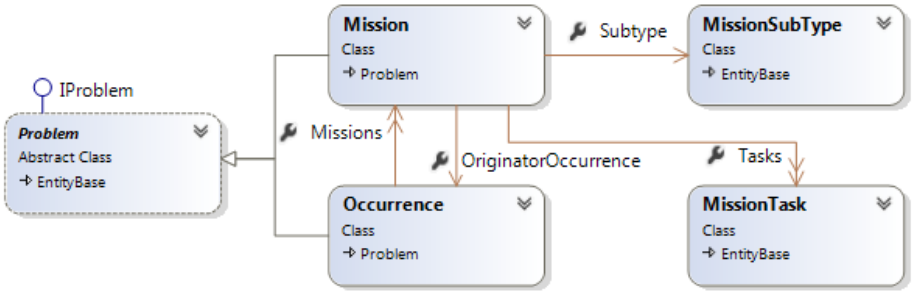


Figura 17 - Entidade Mission.

As dependências de uma missão ao nível do modelo de domínio encontram-se representadas na figura anterior.

As missões podem ser tipo reativo – respostas a ocorrências – ou do tipo proativo – de carácter preventivo. Uma missão pode ter uma ocorrência associada ou então caracterizada

por um subtipo – patrulhamento, escola segura, apoio a lar de terceira idade, etc. Finalmente, uma missão é repartida num conjunto de tarefas que podem ser delegadas a um ou mais agentes. Estas tarefas juntas compõem a missão.

- MissionSubType: Esta entidade representa um subtipo de missão. Este subtipo pode ser patrulhamento, apoio a idosos, escola segura, etc.
- MissionTask: Entidade que representa uma tarefa que pode ser atribuída a agentes para ser executada, tal como é apresentado na figura que se segue:

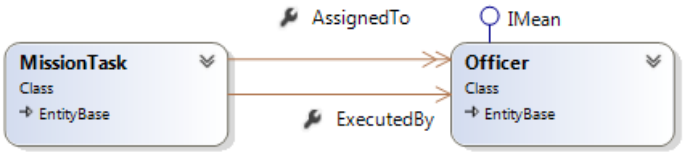


Figura 18 - MissionTaskEntity.

Sendo o modelo composto apenas por objetos POCO, pode ser partilhado entre várias aplicações, sem que nenhuma tenha de incluir bibliotecas adicionais. Desta forma, faz sentido que esteja incorporado num projeto próprio, tendo sido o primeiro projeto implementado no âmbito do SAFER, com o nome de XTraN4ForcesFSDomain.

Este projeto viria, ao longo do tempo a sofrer várias iterações, de forma a adaptar-se melhor às FS.

4.4 Servidor

Uma vez desenhado e implementado o modelo de domínio, o passo seguinte consistiu no desenho da aplicação servidor, para que a aplicação *desktop* pudesse beneficiar de imediato da gestão de entidades remotamente. A aplicação foi construída na forma de um conjunto de *webservices*, cujo projeto foi nomeado SecurityForcesService. A primeira grande decisão a tomar relacionou-se com o tipo de *webservice* a implementar: baseado em *REpresentational state transfer* (REST) [60] ou baseado em *Simple Object Access Protocol* (SOAP) [61]. Após algum estudo sobre o assunto de forma a compreender as vantagens e desvantagens associadas a cada uma das abordagens [62]–[64], optou-se pela implementação de *webservices* baseados numa arquitetura REST. Os principais motivos que levaram a esta decisão consistem na sua maior escalabilidade, na possibilidade de seleção de formato de resposta (o SOAP apenas permite XML – *eXtensible Markup Language*) e conformidade com os pedidos *standard HyperText Transfer Protocol* (HTTP) [65].

Tendo sido decidido optar por *webservices* construídos sobre uma arquitetura REST, o

próximo passo foi decidir que tipo de projeto (no Visual Studio) escolher para a implementação da aplicação. Sendo o objetivo da aplicação a disponibilização de recursos na forma de *Application Programming Interface* (API), a escolha razoável foi a criação de um projeto utilizando a *framework* MVC 4 [66] (em que MVC significa *Model View Control*), tendo sido utilizado o *template* de projeto Web API [67].

Após uma análise à estrutura do projeto, concluiu-se que, por omissão, o Visual Studio incluía um número elevado de referências (bibliotecas), várias das quais eram desnecessárias no contexto desta aplicação. Para colmatar este problema, e começar com um projeto mais "limpo", em vez de remover todas as referências e adicionar depois conforme as necessidades, a solução encontrada foi a instalação de uma extensão para o Visual Studio, denominada de Empty ASP.NET Web API Project Template [68], que permite a criação de um projeto baseado num *template* Web API mais compacto, contendo apenas as referências necessárias. Uma vez criado o projeto, partiu-se para a implementação.

4.4.1 Arquitetura do servidor

Partindo de um projeto baseado em MVC, tal como referimos acima, e sendo MVC por si só uma arquitetura de desenho, é lógico que a arquitetura é composta pelas três camadas do modelo MVC, que determinam uma separação lógica da responsabilidade de processamento e tratamento de informação. Não obstante as camadas do modelo MVC, este servidor conta ainda com uma quarta camada, a de acesso a dados, como ilustra a figura seguinte.

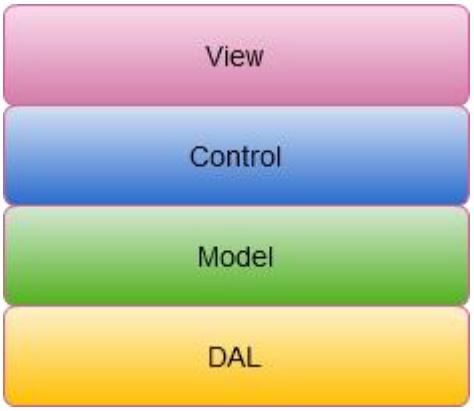


Figura 19 - Arquitetura da aplicação servidora.

Por ordem de apresentação na figura, a arquitetura é composta pelas camadas de apresentação (View), controlo (Control), modelo de dados (Model) e finalmente pela camada de acesso a dados (Data Access Layer – DAL).

A camada de apresentação é composta pelos pedidos HTTP GET, POST, PUT e DELETE – que constituem os canais de entrada de informação no sistema - e pelos dados de retorno.

Tratando-se de uma API, não são disponibilizadas páginas *web*, sendo devolvidos dados via HTTP. Por omissão o sistema devolve dados no formato XML. Contudo, neste projeto o tipo de dados adotado é JavaScript Object Notation (JSON). A adoção de JSON é baseada no facto de este ser um formato leve e mais rápido de processar do que o XML [69], [70].

A camada de controlo é composta por elementos denominados controladores, que respondem com base no *input* (HTTP) a ação a executar sobre o modelo de dados (camada imediatamente abaixo) e determinam que informação deverá ser devolvida à camada acima.

A camada de modelo é composta pelas classes que modelam os dados do sistema, ou modelo de domínio. Neste projeto esta camada está implementada num projeto em separado – XTraN4ForcesFSDomain – como especificado na secção anterior.

Finalmente, a camada de acesso a dados ou DAL é responsável pela persistência e consulta de dados, e é composta por repositórios de dados que comunicam com o ORM, neste caso em particular, NHibernate [71].

De uma forma mais detalhada, é apresentado na figura seguinte o fluxo de informação entre as camadas, para o exemplo de um pedido a uma lista de equipas:

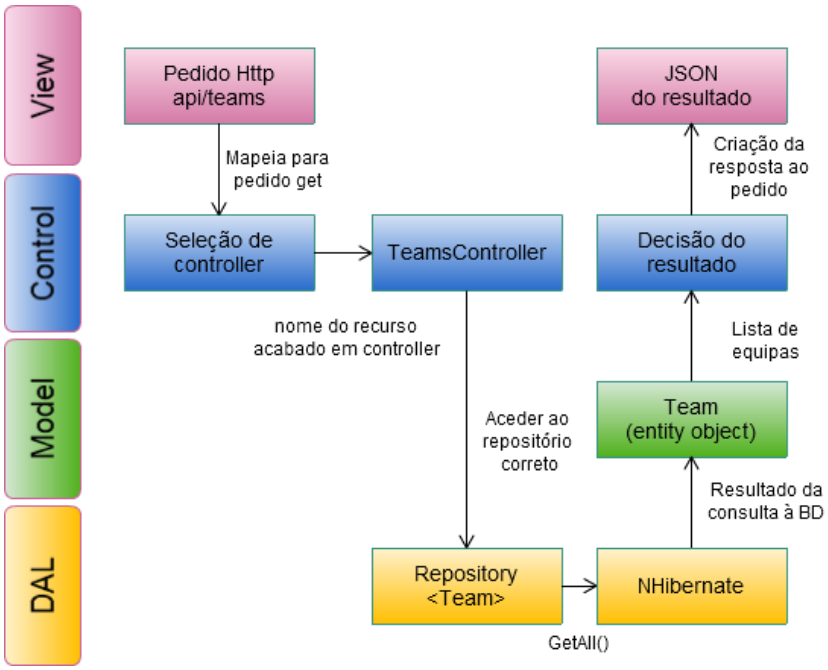


Figura 20 - Fluxo de informação entre componentes da arquitetura.

4.4.2 NHibernate

Na Tecmic é política utilizar ferramentas de ORM para abstrair o modelo de domínio da forma como os dados são persistidos. A ferramenta utilizada é denominada de NHibernate, sendo utilizada em vários sistemas da Tecmic, assim como em vários projetos no 4 Forces. Faz sentido utilizar a mesma ferramenta de forma a manter a conformidade tecnológica, assim

como beneficiar do conhecimento dentro da empresa sobre a ferramenta.

O projeto NHibernate é uma adaptação de Java para .Net da *framework* Hibernate [72] e está disponível para *download* de vários modos, sendo o preferido para o SAFER através do gestor de pacotes NuGet [73], porque este método permite atualização das bibliotecas de forma automática a partir do próprio Visual Studio diretamente no projeto, sem ter de se estar manualmente a remover e adicionar as versões atualizadas. O NHibernate é um projeto maduro, bem documentado e com excelente suporte por parte da comunidade, que permite tornar agnóstica a comunicação com um sistema de gestão de base de dados – DBMS (Database Management System) – desde que este seja suportado. São suportados a maioria dos principais DBMS no mercado, incluindo Microsoft SQL Server [74] (2000/2005/2008), Oracle [75], Microsoft Access [76], Firebird [77], PostgreSQL [78], DB2 UDB [79], MySQL [80] e SQLite [81]. Nos nomes apresentados SQL significa Structured Query Language, a liguagem utilizada para realização de consultas aos dados num DBMS.

Durante o desenvolvimento deste projeto o NHibernate foi configurado para utilizar SQLite, pois deste modo a criação da base de dados é realizada de modo automático – o ficheiro que contém a base de dados é regenerado automaticamente, enquanto que de outro modo seria necessária intervenção manual (a título de exemplo, tipicamente, o *user* disponibilizado para a aplicação tem apenas permissões de acesso, leitura e escrita e não de alteração de tabelas) Mais tarde, quando o sistema estiver em produção, está prevista a utilização de SQL Server 2008, que é utilizado também em outros projetos no 4 Forces. O NHibernate integra facilmente com a biblioteca de *Logging* log4net [82], biblioteca de eleição na Tecmic para escrita de *logs*.

O NHibernate é a causa da *keyword virtual* nas entidades do modelo de domínio, devido ao seu modo de funcionamento através da criação de *proxy objects* de forma a implementar *lazy loading*. Explicando de uma forma simplificada, o NHibernate permite efetuar *lazy loading* de entidades, de um modo transparente, sem necessidade de ter as entidades do modelo a estender de uma "NHibernate *base class*" ou a implementar uma interface específica. Por omissão este comportamento está ativado, sendo no entanto configurável. *Lazy loading* é o processo que permite, ao aceder a um objeto, que este não seja retirado da base de dados por completo, ou seja, apenas a informação relevante é obtida. À medida que se trabalha sobre a entidade o NHibernate acede à informação apenas no momento em que esta é realmente necessária. Exemplificando, a entidade equipa (Team no modelo de domínio) tem uma propriedade Unit (do tipo TeamUnit). Ao trabalhar sobre um objeto do tipo Team, mas sem aceder à propriedade Unit, o NHibernate não carrega a informação da unidade se esta não for

necessária, sendo que só irá aceder à mesma quando for executada uma ação sobre a propriedade. Este processo só é possível devido ao uso da *keyword virtual*. O que acontece (de forma transparente) é que o NHibernate cria subclasses (*proxies*) das entidades e faz *overriding* dos métodos e propriedades. No mesmo exemplo, ao ser acedida a propriedade Unit, o objeto *proxy* interceta essa chamada, verifica se existe valor e se não existir, vai aceder à informação na hora, procedendo à chamada da implementação original (aceder à propriedade) de seguida. É este comportamento que obriga a que todas as entidades no modelo tenham as suas propriedades e métodos declarados como *virtual*. Caso seja desativado o comportamento de *lazy loading*, não é necessário que a *keyword virtual* esteja presente. Contudo, por motivos de desempenho, é fortemente encorajado deixar ativo o *lazy loading* e, neste projeto, está ativo. Apesar desta restrição, os objetos do modelo de domínio não deixam de ser POCO uma vez que não estão dependentes de nenhum *Attribute* específico e não necessitam de estender de uma classe ou implementar nenhuma interface em particular do NHibernate para funcionar.

Para operar sobre o NHibernate são necessários dois passos essenciais: mapeamento de entidades e configuração inicial. O NHibernate não tem capacidade para decidir o tipo de mapeamentos entre entidades. Não sabe qual a propriedade que tem de utilizar como chave primária, não sabe relações de "N-M" para gerar as tabelas nem restrições (*constraints*) sobre os valores a inserir. É por isso necessário mapear as propriedades das entidades, para que o NHibernate consiga entender as suas relações e restrições. Tradicionalmente, os mapeamentos são realizados com recurso a ficheiros do tipo XML, tipicamente um por entidade. Por exemplo, um ficheiro para mapear a entidade Team teria o nome Team.hbm.xml. Este modo de configuração caiu em desuso, porque apresenta sérios inconvenientes, entre os quais a quantidade de código XML que tem de ser escrito, o facto de ser um procedimento aborrecido, repetitivo e que pode dar origem a várias falhas, que só são detetadas quando a aplicação já está a correr. De igual modo, ao renomear, adicionar ou remover um campo de uma entidade, é necessário alterar o ficheiro XML, uma operação que é facilmente esquecida e bastante propícia a erros. Para colmatar este problema, existem dois projetos (bibliotecas) principais que permitem mapear as entidades sem recurso a XML: NHibernate.Mapping.Attributes [83] e Fluent NHibernate [84]. A primeira opção é utilizada em vários produtos Tecmic, e até em alguns projetos no 4 Forces. O modo de funcionamento consiste em decorar com *Attributes* o modelo de domínio, onde é possível especificar, sem recurso a XML, as várias configurações que a entidade deve sofrer ao ser tratada pelo NHibernate. Por ser uma biblioteca utilizada em larga escala na Tecmic, foi considerada a sua

utilização também para o SAFER, de modo a manter a coerência entre bibliotecas utilizadas. Porém, estar a decorar todas as entidades com atributos específicos do NHibernate iria contra o princípio do POCO, estando a "contaminar" o domínio. Por isso, foi utilizada a biblioteca Fluent NHibernate, que se viria a revelar numa verdadeira mais-valia para o projeto, simplificando muito o processo de configuração inicial e mapeamento do NHibernate.

A melhor forma de descrever o projeto Fluent NHibernate (FNH) é com as palavras do próprio *site*: "Fluent, XML-less, compile safe, automated, convention-based mappings for NHibernate" [84]. Colocando estas palavras em português, FNH é, em primeiro lugar, uma biblioteca de mapeamentos para o NHibernate. A sua utilização é *fluent*, (um estilo de código orientado à melhoria da legibilidade do código escrito [85]), permitindo a compreensão do código através do encadeamento lógico de operações. Funciona sem recurso a XML, sendo *compile safe*, visto que, por exemplo, ao renomear uma propriedade ou uma entidade no modelo de domínio, as alterações são refletidas de imediato, sem necessidade de alterar ficheiros manualmente, sendo os erros de mapeamento detetados pelo compilador e não sendo necessário esperar pela altura de *runtime* para saber se os mapeamentos estão corretamente escritos ou não. E, por último mas não menos importante, é uma biblioteca automatizada e baseada em convenções. O FNH utiliza o princípio de *convention over configuration* [86], [87], simplificando drasticamente o processo de configuração através da tomada de decisões automática com base em convenções e normas reconhecidas. Este processo de tomada de decisões automático pode parecer redutor, restringindo o processo de configuração, mas tal não sucede. Todos os comportamentos automáticos podem ser configuráveis manualmente consoante as necessidades. O FNH toma simplesmente a liberdade de executar determinadas ações por omissão de forma a adaptar-se à maioria das situações sem intervenção manual.

Para mapear as entidades, o FNH disponibiliza dois tipos de mapeamentos: *fluent mappings* e *auto-mappings*. Os *fluent mappings* permitem mapear através de código o equivalente ao tradicional XML, em que o programador define manualmente as restrições e os mapeamentos, mas com a vantagem de ser em código *strongly typed*, permitindo alterações através de *refactoring* sem ter de alterar *hardcoded strings*. Porém, a funcionalidade que mais atraiu no FNH foi a de *auto-mapping*. Com base no princípio *convention over configuration*, o FNH permite mapear todo o modelo de domínio com recurso a poucas linhas de código, sem a necessidade de escrita de qualquer mapeamento. Através do uso de *Reflection* [88] o FNH inspeciona o modelo de domínio e decide que tipos deve mapear e como. Um exemplo de convenção é o mapeamento da chave primária da tabela na base de dados. Tendo uma entidade com uma propriedade denominada Id, o FNH mapeia automaticamente essa

propriedade como chave primária. Assim, a funcionalidade de *auto-mapping* reduz complexidade e delega a responsabilidade do mapeamento por parte do programador para a biblioteca que trata disso de forma automática, bastando apenas indicar ao FNH onde (em que *assembly*) deve procurar as entidades.

Além do mapeamento, e como já referido, a biblioteca FNH permite realizar também a configuração inicial do NHibernate, processo outrora realizado em XML. A configuração é apresentada na Figura 21.

O código listado na figura é um excerto da classe NhSessionFactory que é abordada com mais detalhe mais à frente no documento. Para efeitos de demonstração da biblioteca, apenas dois métodos são apresentados: o construtor da classe e o método "DecideConfiguration" para obter uma configuração. O construtor inicia toda a configuração do NHibernate com recurso à biblioteca FNH, cujo processo pode ser compreendido em quatro passos: obtenção de uma configuração de persistência, criar um modelo de persistência, configuração e obtenção de um objeto ISessionFactory.

O primeiro passo é realizado com recurso ao método DecideConfiguration. Com base num parâmetro que indica se deve devolver uma configuração para SQLite ou para SQL Server, é obtida uma configuração de persistência, utilizando a biblioteca FNH. No caso de SQL Server a configuração é realizada com recurso a uma *connection string* enquanto que ao escolher SQLite a configuração é criada com base no ficheiro a utilizar (apesar de ser possível também configurar com uma *connection string*). De seguida, é indicado o modelo de persistência. Neste ponto é necessário indicar ao FNH o *namespace* onde deverá procurar as entidades do modelo de domínio para as mapear automaticamente. Uma vez que a composição do modelo de domínio inclui classes abstratas, é necessário indicar quais as classes abstratas que deve considerar como entidades e excluir as que são apenas base para outras classes. Para o efeito, o FNH disponibiliza os métodos IgnoreBase e IncludeBase. Por último, são modificados alguns comportamentos por omissão do FNH, quer seja para não mapear propriedades das entidades que sejam apenas de leitura – possuem *getter* mas não possuem *setter* – quer seja para adicionar comportamento de *cascade* ao persistir – neste caso deve alterar não só a entidade em si como também as propriedades que contiverem entidades alteradas.

```
private NhSessionFactory()
{
    IPersistenceConfigurer databaseConfiguration = DecideConfiguration(true);
    var autoPersistenceModel = AutoMap.AssemblyOf<EntityBase>()
        .Where(t => t.Namespace.EndsWith("Domain"))
        .IgnoreBase<EntityBase>()
        .IgnoreBase<Problem>()
        .IncludeBase<Mission>()
        .IncludeBase<Mean>()
        .Conventions.Add(DefaultCascade.All())
        .OverrideAll(x => x.IgnoreProperties(prop => ! prop.CanWrite));

    var fluentConfiguration = Fluently.Configure()
        .Database(databaseConfiguration)
        .Mappings(m =>
            {
                m.AutoMappings.Add(autoPersistenceModel);
                //.ExportTo(@"C:\xmlmappings");
            })
        .ExposeConfiguration(cfg =>
            {
                new SchemaUpdate(cfg).Execute(sqlText =>
                    {
                        var updateFile = HttpContext.Current.Server
                            .MapPath("~/App_Data/Update.sql");
                        using (var file = new FileStream(
                            updateFile, FileMode.Append, FileAccess.Write))
                        using (var sw = new StreamWriter(file))
                        {
                            sw.Write(sqlText);
                            sw.Close();
                        }
                     }, true);
                 cfg.SetInterceptor(new NhInterceptor());
            });
    var config = fluentConfiguration.BuildConfiguration();
    _sessionFactory = config.BuildSessionFactory();
}

private static IPersistenceConfigurer DecideConfiguration(
    bool useSQLite = true)
{
    if (useSQLite)
    {
        var sqLiteCfg = SQLiteConfiguration.Standard
            .UseReflectionOptimizer()
            .UsingFile(HttpContext.Current.Server.MapPath(
                "~/App_Data/SecurityForces.db3"));
        return sqLiteCfg;
    }

    var msSql2008Cfg = MsSqlConfiguration.MsSql2008
        .UseReflectionOptimizer()
        .ConnectionString(WebConfigurationManager
            .ConnectionStrings["NHibernateConnString"].ConnectionString);
    return msSql2008Cfg;
}
```

Figura 21 - Configuração do NHibernate usando a biblioteca Fluent NHibernate.

O terceiro passo consiste na configuração propriamente dita. São indicados ao FNH a configuração e modelo de persistência a aplicar, usando os objetos obtidos anteriormente. O código comentado (a verde) indica que deverão ser exportados os ficheiros .hbm.xml para uma pasta. Esta opção é útil para verificar que os mapeamentos estão a ser realizados de forma correta. Opcionalmente é exposta uma configuração que permite gerar um *script* SQL ao ser detetada uma alteração ao modelo de domínio, de modo a poder atualizar a base de dados existente sem ter que recriar a mesma do zero. É registado ainda um *Interceptor* cuja finalidade é detalhada mais à frente neste documento.

Por último, a configuração é construída e finalmente basta criar uma *ISessionFactory*, a partir da qual são criadas sessões (objetos *ISession*) do NHibernate, permitindo realizar consultas e operações Create, Replace, Update e Delete (CRUD) sobre o modelo de dados.

O processo de configuração é um processo que, envolvendo *Reflection*, não é um processo computacionalmente leve, uma vez que o FNH tem de percorrer todas as entidades do modelo de domínio e gerar a base de dados se necessário. Por isso, deve apenas ser executado uma vez durante o ciclo de vida da aplicação. De forma semelhante, a criação da *ISessionFactory* é um processo computacionalmente pesado [71], pelo que só deve ser criada uma instância desse objeto (também conhecido como padrão de desenvolvimento *Singleton* [89]). A título experimental, caso a configuração do NHibernate não seja reutilizada, e se forem obtidas novas configurações (assim como *session factories*) em cada pedido, o tempo que o pedido demora a ser executado aumenta de uma fração de segundo para aproximadamente 10 segundos, o que demonstra que estes objetos só devem realmente ser criados uma vez no ciclo de vida da aplicação e reutilizados para todos os pedidos.

Para resolver este problema, a criação destes dois objetos únicos foi obtida através do encapsulamento numa classe, denominada NhSessionFactory, de forma a garantir que apenas uma instância é criada, implementando o padrão *Singleton*.

A partir da NhSessionFactory é possível fazer múltiplos pedidos para obter uma ISessionFactory, obtendo sempre a mesma instância. Esta *ISessionFactory* é depois introduzida nos *Controllers* (camada de controlo) utilizando técnicas de *Dependency Injection* [90], como é abordado mais à frente neste capítulo.

4.4.3 HTTP, Routing e Controllers

Os controladores (*controllers*) são as classes que lidam com os pedidos HTTP ao servidor, traduzindo os pedidos em ações. A *framework* ASP.NET possui mecanismos internos que permitem mapear os pedidos para o controlador apropriado. Num projeto Web API esse

mecanismo é estendido de forma a englobar o tipo de pedido (GET, POST, PUT ou DELETE), que é mapeado automaticamente para a ação correspondente no controlador, partindo do princípio *convention over configuration*.

Por omissão, um pedido ao endereço [endereço do servidor]/api/teams realizado via GET é mapeado para um *controlador* com o nome "TeamsController". Tal acontece devido à convenção utilizada para nomear os controladores. Caso não exista uma classe que herde de *ApiController* com esse nome, o servidor rejeita o pedido com o código 404 – recurso não encontrado. Uma vez que o pedido é realizado via GET, o sistema procura um método cujo nome comece por Get como, por exemplo, GetAllTeams() e executa o mesmo. Caso sejam especificados parâmetros no endereço, o sistema procura um método que receba o mesmo número de parâmetros, executando-o automaticamente. Utilizando o mesmo exemplo, um pedido via GET ao endereço [endereço do servidor]/api/teams/5, resultaria na execução do método GetTeamById(int id). Mais uma vez, o nome do método não é importante, desde que comece pelo tipo de pedido (GET, POST, PUT, DELETE). Assim, continuando com o exemplo anterior, se a classe TeamsController definir os métodos que constam da tabela seguinte, estes são mapeados automaticamente:

Tabela 4 - Mapeamento de pedidos Web API

Método	Descrição
Get()	Pedido GET. Utilizado para obter todas as equipas.
Get(Guid id)	Pedido GET. Obtém a equipa com o id especificado.
Post(Team team)	Pedido POST. Insere a equipa no sistema.
Put(Team team)	Pedido PUT. Atualiza a equipa no sistema.
Delete(Guid)	Pedido DELETE. Apaga a equipa com o id especificado do sistema.

Esta tabela contém a listagem dos métodos implementados na classe TeamsController. Para as restantes entidades (utilizadores, agentes, ocorrências, equipamentos, etc.) os métodos seguem o mesmo padrão de nomes, como é detalhado mais à frente neste documento. Note-se que existem dois pedidos Get, mas com um número de parâmetros diferente. Tal é perfeitamente válido, uma vez que o parâmetro – que varia consoante o pedido: id, team – é opcional.

Esta convenção oferece a vantagem óbvia de ter uma rota por omissão para a qual os pedidos são mapeados para ações automaticamente, tendo porém a desvantagem de não permitir dois métodos com o mesmo número de parâmetros, como por exemplo GetTeamById(int id) e

GetTeamByName(string name). Uma vez que para esta rota apenas parte do nome do método e o número de parâmetros são contabilizados, o sistema não sabe qual dos métodos deve executar, lançando uma exceção alertando para várias ações possíveis para o mesmo pedido.
Por outro lado, as rotas podem ser configuradas de acordo com as necessidades e o gosto de quem desenvolve as aplicações. Uma abordagem bastante popular para quem necessita de permitir vários métodos com o mesmo número de parâmetros a executar é a configuração de rotas baseadas no nome da ação a executar. Com esta abordagem, o endereço do pedido torna-se mais comprido, adicionando porém mais informação visual (endereço mais detalhado), o que por si só pode ser um indício de usabilidade melhorada, pois quem acede tem mais informação visual sobre o pedido [91], [92]. Isto é válido tanto para *sites*, em que através de um endereço bem estruturado é possível o utilizador saber situar-se no *site*, como para APIs, em que um pedido bem estruturado dispensa parte da documentação. Um exemplo disto será um pedido [endereço do servidor]/api/teams/eq01. Sem mais informação detalhada, apenas se sabe que se está a aceder a uma equipa e que "eq01" contém informação relevante sobre a mesma. Pode, no entanto, dar origem a dúvidas por parte de quem acede ao endereço acerca do que está a acontecer, pois não é claro se "eq01" é um Id, nome ou outro identificador qualquer. Com a abordagem de rota baseada no nome da ação tal não sucede, desde que o nome da ação seja bem explícito. Um exemplo alterado poderia ser [endereço do servidor]/api/teams/getByName/eq01, em que o endereço não deixa margem para dúvidas de que se está a procurar uma equipa pelo nome. Esta abordagem permite ainda ter vários métodos com o mesmo número de parâmetros, o que é bastante atrativo em certas circunstâncias. Contudo, é necessário especificar o tipo de acesso ao método (GET, POST…), pois o sistema deixa de ter forma de mapear o tipo de pedido de forma automática.
De seguida é apresentada uma tabela resumida com prós e contras da utilização de rota por omissão *versus* rota mapeada por ações:

Tabela 5 - Comparação entre abordagens de *routing*.

	Rotas por omissão	Rotas por ação
Mapeamento automático de tipo de pedido	Sim. GET, POST, PUT e DELETE	Não. É necessário especificar manualmente.
Permite alterar o nome do método sem alterar o cliente	Sim. Desde que este comece por Get, Post, Put ou Delete	Não. O nome da ação muda o endereço a aceder.

Permite vários métodos com o mesmo número de parâmetros.	Não. O sistema não sabe qual o método que tem de executar.	Sim.
Informação detalhada no pedido.	Não. O esquema da rota é limitado.	Sim. O nome pode providenciar informação descritiva sobre o método que executa.

Como é possível comprovar, nenhuma das abordagens é perfeita. Ambas possuem vantagens e desvantagens. Por isso, para a aplicação implementada optou-se por uma abordagem híbrida. Para os controladores "normais" – os que são utilizados para mapear pedidos de aplicações cliente – utiliza-se o sistema de rotas por omissão. A lógica da parte do cliente é simplificada, pois é mais fácil à partida saber qual o endereço a aceder para cada entidade/controlador. Contudo, o sistema dispõe ainda de controladores de caráter administrativo que suportam várias funcionalidades e, com isso, requerem vários métodos com o mesmo número de parâmetros. Os pedidos para estes controladores são mapeados com base no nome da ação a executar em vez de utilizar a rota por omissão (que limitaria o número de métodos disponibilizados num só controlador).

4.4.4 Dependency Injection

Dependency Injection (DI) é um padrão de desenvolvimento de software que deriva do padrão Inversion of Control (IoC), um padrão que permite desacoplar referências e tarefas computacionais da sua implementação [90]. DI é utilizado, por exemplo, para juntar componentes tipicamente implementadas por diferentes equipas, em que o único fator conhecido *a priori* é a dependência de uma determinada abstração comum (tipicamente uma interface) A por parte de um componente B. O componente B apenas sabe que necessita da abstração A para poder funcionar.

No caso de projetos MVC e/ou Web API, a utilização de DI é bastante comum. O cenário típico: quando um pedido chega ao servidor (voltando ao exemplo das equipas) para o endereço [endereço do servidor]/api/teams/, a responsabilidade é delegada paro o controlador apropriado, a classe TeamsController. Através do processo automático de mapeamento, é executado o método Get() que devolve as equipas do sistema. Assim, o controlador tem de obter uma sessão do NHibernate e efetuar a pesquisa, devolver os dados e fechar a sessão. Tipicamente o controlador não trata deste processo diretamente. A única função do

controlador é executar ações e devolver dados, não tendo conhecimento de como os dados são acedidos. O acesso à camada DAL é efetuado por um repositório que conhece efetivamente a camada DAL e sabe como utilizar a mesma, de acordo com a imagem seguinte:

Figura 22 - Esquema simplificado de dependências de um controlador.

A imagem anterior não reflete a implementação real, simplificando apenas o problema em questão: dependências de um *controlador*.

Fica então claro que o TeamsController necessita de um repositório de equipas, que por sua vez necessita de uma sessão do NHibernate. Uma primeira, e má, abordagem ao problema poderia consistir em incluir um repositório na classe TeamsController e abrir uma sessão no próprio repositório, fechando-a de seguida. Esta abordagem tem falhas sérias, mesmo que não se considere a não separação de responsabilidades (o controlador ficaria a conhecer a camada de persistência diretamente através do repositório). Em primeiro lugar, esta abordagem iria falhar se fosse necessário mais do que um pedido à base de dados (por exemplo, um método de autenticação poderia, para além de aceder à informação de um utilizador, querer modificar a data da última autenticação que poderia ter de ser num repositório à parte), pois se cada pedido ao repositório abrisse a sua própria sessão, uma transação não serial possível ao trabalhar com vários repositórios ao mesmo tempo. Em segundo lugar, a alteração numa das componentes (controlador ou repositório) iria alterar seriamente as componentes adjacentes, o que não é desejável. Uma alternativa seria passar no construtor do controlador um objeto que implementasse uma interface para o repositório contendo apenas a definição dos métodos necessários, o que iria, pelo menos, remover o conhecimento do controlador sobre a camada de DAL. Esta solução, infelizmente, não é possível. Tal acontece porque a plataforma ASP.NET é responsável por instanciar os controladores, através de um construtor sem parâmetros, não permitindo controlo sobre a forma como o controlador é instanciado. Utilizando DI já é possível, por exemplo, ter construtores de controladores contendo argumentos, uma vez que a instanciação dos controladores passa a ser da responsabilidade de DI e não da plataforma ASP .NET. Este processo envolve a utilização de um *provider* (ou *container*) que permite registar objetos e as suas dependências em *runtime*. São várias as bibliotecas existentes que disponibilizam *containers*, entre as quais StructureMap [93], Unity

[94], Castle Windsor [95], Ninject [96] e AutoFac [97]. Neste projeto foi utilizado o StructureMap visto ser um projeto maduro que dispõe de um bom suporte por parte da comunidade.

A utilização do StructureMap permitiu, de uma forma simples e concisa, registar as dependências entre os vários módulos, e estas são injetadas em *runtime* nos componentes. Uma outra grande vantagem da utilização de DI consiste na gestão do ciclo de vida (*scope*) dos objetos, sendo o caso flagrante o da sessão do NHibernate. É boa prática recorrer a uma transação quando se lida com o NHibernate. Esta transação é tipicamente encapsulada numa unidade de trabalho, de acordo com o padrão de desenvolvimento *Unit Of Work*. Esta unidade de trabalho abre uma transação para cada pedido (*session per request* [98]), utilizando uma sessão do NHibernate [99]. É possível definir automaticamente que o *scope* de determinados objetos é por pedido. Cada pedido ao servidor resulta numa nova sessão e, consequentemente, numa nova transação. Na Figura 23 é apresentado o diagrama que representa as dependências entre os vários objetos.

Os controladores "normais" são aqueles que são utilizáveis para os clientes dos *webservices*, disponibilizando ações sobre utilizadores, agentes, equipas, equipamentos, ocorrências, missões, etc. Estes estendem a classe ControllerBase que providencia um controlador genérico (abordado em detalhe mais à frente). Ao mesmo nível hierárquico do ControllerBase estão controladores de caráter administrativo e o controlador de *login*. Subindo na hierarquia dos controladores, encontra-se a classe ControllerBaseUow. Esta classe é decorada com um *attribute* criado denominado UnitOfWorkAttribute, indicando que ações deste controlador (e suas subclasses) são encapsuladas numa unidade de trabalho – IUnitOfWork. É no *attribute* que ocorre a gestão da transação: abertura e encerramento. Os controladores não acedem diretamente à DAL, sendo tal responsabilidade delegada para os repositórios. Para que os controladores possam aceder aos dados, necessitam de um ou mais repositórios (IBaseRepository), os quais são injetados nos controladores.

Note-se que os controladores não conhecem a implementação dos repositórios, apenas a interface que os mesmos disponibilizam. A implementação (BaseRepository) é registada utilizando DI e só o *container* de DI tem conhecimento da implementação dos repositórios utilizada. Isto permite que a implementação conheça a DAL, mas que para os controladores seja um processo totalmente transparente. A implementação necessita de uma unidade de trabalho (IUnitOfWork) para poder trabalhar sobre as entidades.

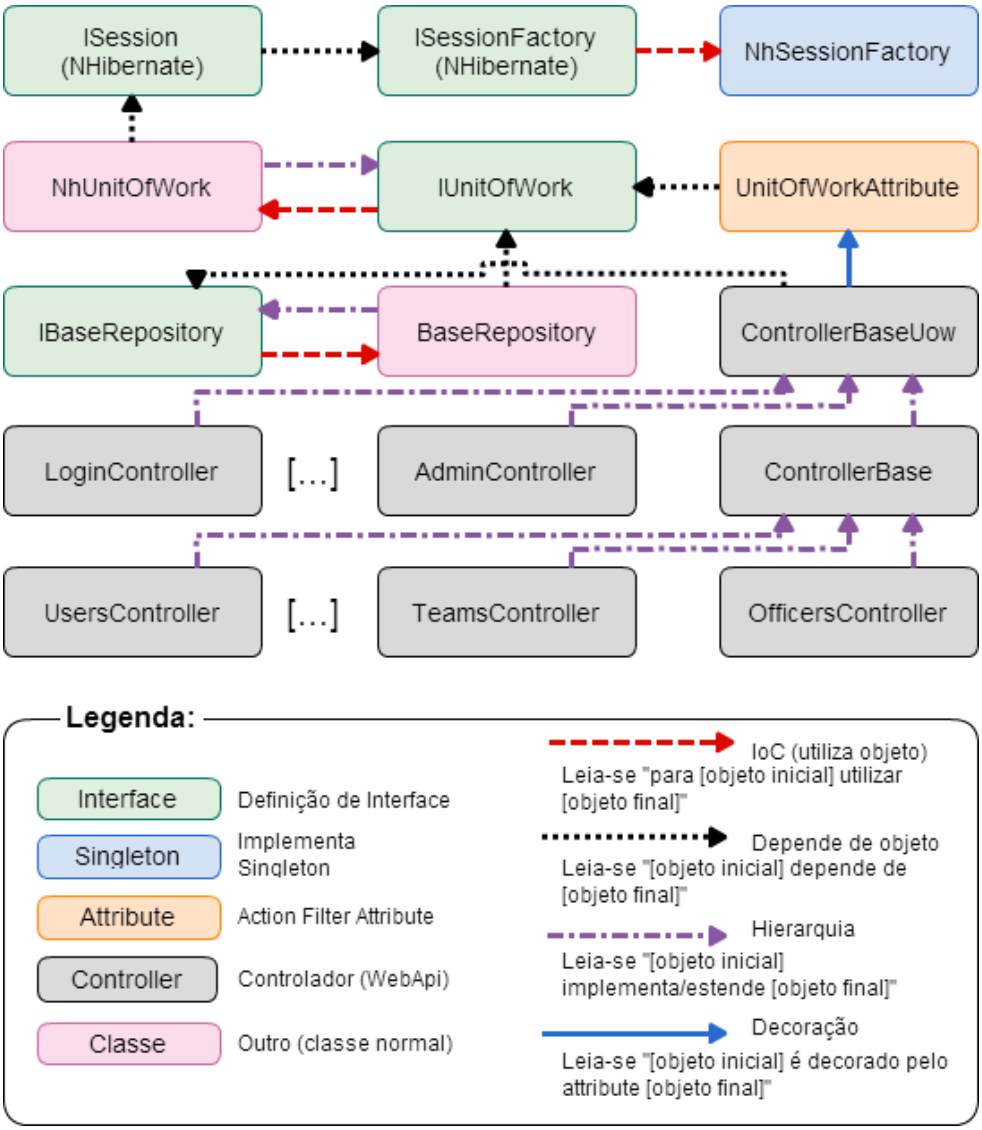


Figura 23 - Diagrama de dependências no servidor.

Mais uma vez, a implementação da unidade de trabalho (NhUnitOfWork) é transparente, sendo o *scope* delimitado a cada pedido. Isto significa que a instância do objeto IUnitOfWork utilizado pelos repositórios é a mesma utilizada pelo *attribute* em cada pedido. Caso contrário, seriam utilizadas duas sessões no mesmo pedido, o que resultaria em incoerência ao persistir os dados. Finalmente, a implementação da unidade de trabalho necessita de uma sessão do NHibernate, que provém de uma *session factory*, registada usando DI. É este o esquema de dependências na aplicação.

4.4.5 Unit of work

A correta implementação do padrão de desenvolvimento *Unit of Work* (UoW) desempenha um papel importante na aplicação. É a UoW que encapsula e gere a transação do Nhibernate e, consequentemente, de acesso à base de dados. A interface disponibilizada – IUnitOfWork –

disponibiliza os seguintes métodos/propriedades: CurrentSession, Begin(), Commit() e Rollback(). A implementação é realizada pela classe NHunitOfWork e injetada usando DI nos repositórios e no *attribute* que gere a unidade de trabalho (UnitOfWorkAttribute). O *attribute* consiste numa classe que estende *ActionFilterAttribute*, que interseta a execução das ações dos controladores, iniciando a unidade de trabalho antes e finalizando quando a ação termina, com *commit* ou *rollback*, respetivamente, em caso de sucesso ou erro. Durante uma ação um ou mais repositórios podem ser utilizados para gerir entidades. A unidade de trabalho utilizada é a mesma que foi iniciada no *attribute*, utilizando portanto a mesma transação. Esta abordagem permite operações que alterem entidades em vários repositórios, salvaguardado a transação no final em caso de sucesso ou desfazendo todas as alterações em caso de erro. Deste modo a consistência dos dados é assegurada para cada pedido.

4.4.6 Repositórios

A abstração da DAL é realizada através de repositórios, uma adaptação do padrão de desenvolvimento *Repository* [100]. Pelas palavras de M. Fowler sobre o repositório, "a Repository mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects." [101]. Um repositório constitui uma abstração do acesso a dados de forma genérica, disponibilizando acesso aos dados como se se tratasse de uma coleção de objetos em memória. Tal significa que o acesso a dados está encapsulado sob uma interface única, não sendo relevante a fonte dos dados (bases de dados, outros *webservices*, ficheiros, etc.). Esta abordagem permite que os objetos que requerem a informação possam pesquisar sobre os dados facilmente utilizando as técnicas que a plataforma .NET disponibiliza, dando ênfase especial à Language-Integrated Query (LINQ) sem ter conhecimento do ORM ou outra qualquer forma de acesso a dados. No SAFER a interface que representa um repositório é IBaseRepository, cuja definição se encontra na figura seguinte:

```
public interface IBaseRepository<TEntity, TId>
{
    IEnumerable<TEntity> GetAll(bool includeRemovedItems = false);

    IEnumerable<TEntity> GetWhere(Func<TEntity, bool> condition,
        bool includeRemovedItems = false);

    TId Insert(TEntity entity);

    TEntity Update(TEntity entity);

    bool Delete(TId id);
}
```

Figura 24 - Interface IBaseRepository.

Esta interface foi planeada para ser genérica, aceitando dois tipos de dados genéricos: tipo de entidade e tipo de chave primária (ou tipo de Id). Deste modo, a interface adapta-se a qualquer entidade do modelo de domínio e, se por ventura se decidir mudar o tipo de Id das entidades (digamos de *Guid* para *int* ou *long*), não é necessário alterar o repositório ou a sua implementação.

A Figura 24 apresenta uma versão simplificada da interface IBaseRepository. A interface disponibiliza métodos que permitem realizar a maioria de operações típicas sobre os dados, incluindo obter todos os elementos, obter elementos que satisfaçam determinada condição, obter um objeto pelo seu identificador, verificar se existe algum elemento que satisfaça uma condição, adicionar novo elemento, e atualização ou remoção de um elemento existente.

De notar também que os métodos de pesquisa sobre os dados têm um parâmetro adicional, "includeRemovedItems", que permite especificar se deverão ser incluídos dados de elementos removidos ou não na pesquisa, sendo este comportamento desativado por omissão.

Este comportamento é justificável pelo ambiente empresarial em que se insere o projeto relativamente à remoção de dados. Uma das decisões tomadas foi a de não apagar os objetos da base de dados a partir do *webservice*, de modo a evitar a perda acidental de dados importantes (e mais tratando-se de um contexto relacionado com Forças de Segurança). Os dados são apenas marcados como removidos, não sendo obtidos em futuras consultas. O parâmetro opcional permite a implementação deste comportamento. Sendo uma base de dados com um potencial volume de dados elevado, está previsto um mecanismo de limpeza de dados da base de dados (por exemplo a cópia de dados removidos para uma base de dados separada) de modo a impedir que esta cresça de forma descontrolada. Este mecanismo está fora do âmbito deste projeto, ficando para trabalho futuro.

Como foi já referido, a utilização de DI simplifica o processo de obtenção de repositórios, injetando em *runtime* nos controladores uma instância de um objeto que implemente esta interface. A interface desconhece por completo o modo de acesso a dados, enquanto que a classe que contém a implementação (BaseRepository) já conhece o objeto de unidade de trabalho à qual acede para obter a sessão do NHibernate, permitindo a modificação e consulta dos dados. Com este processo é possível injetar a implementação correta do repositório nos controladores, sendo transparente para os mesmos toda a camada DAL.

4.4.7 Controladores

Os controladores (*controllers*) são o ponto de entrada no sistema. Pedidos efetuados ao servidor são mapeados automaticamente para o controlador apropriado e, caso não haja um

controlador capaz de mapear o pedido, o mesmo é rejeitado com o código 404 – recurso não encontrado.

O mapeamento dos pedidos para os controladores e ações correspondentes é realizado de forma automática de acordo com o processo já explicado anteriormente. Dado que todos os controladores implementados necessitam de obter dados armazenados ou de efetuar operações CRUD sobre os mesmos, é necessário que aqueles sejam geridos por uma unidade de trabalho, abrindo uma ligação para o NHibernate antes de executar a ação do controlador e finalizando a mesma (*commit*/*rollback*) no final da ação executada. Esta gestão automática é realizada decorando o controlador com o *attribute* desenvolvido para o efeito, o UnitOfWorkAttribute. Por este motivo implementou-se um controlador base, denominado ControllerBaseUow que deriva dos controladores de sistema (ApiController). Este controlador apenas contém a decoração com o atributo, servindo como base para os restantes controladores implementados.

Ao desenvolver os *webservices* para o SAFER, começou-se por implementar um controlador para *login* no sistema, cuja função é validar a existência do utilizador, devolvendo a informação do mesmo em caso de sucesso. Uma vez definida a forma de autenticar utilizadores, implementaram-se os restantes controladores do sistema. Em paralelo com os controladores "normais" – que são desenhados para mapear os pedidos de aplicações cliente – foram implementados dois controladores de caráter administrativo – para facilitar a gestão e testar a aplicação.

Os controladores administrativos são o AdminController e o DummyController. Estes controladores foram implementados com recurso a vários métodos com igual número de parâmetros, sendo os pedidos mapeados pelo nome da ação ao invés das regras por omissão da *framework* Web API.

O controlador AdminController possui funcionalidades para criar utilizadores por omissão, tipos de ocorrência e dados geográficos. Os tipos de ocorrência por omissão são provenientes de um ficheiro de modo a permitir a funcionalidade de *auto-complete* quando se insere uma nova ocorrência na aplicação cliente. O mesmo procedimento é utilizado para carregar a informação relativa a pontos de interesse relevantes para a esquadra com a finalidade de poderem ser representados no mapa.

No que diz respeito a dados geográficos, para além de pontos de interesse, são também inseridos por omissão dados relativos aos nomes de freguesias, concelhos e distritos de Portugal, incluindo (opcionalmente) dados das regiões autónomas da Madeira e dos Açores. Estes dados são provenientes da Carta Administrativa Oficial de Portugal (CAOP) [102],

sendo a versão em vigor na altura da escrita deste documento a CAOP 2013.

O DummyController foi concebido para auxiliar nos testes do sistema. Durante a fase de implementação (tanto a nível de servidor como a nível *desktop*) é comum, devido a erros de implementação, inserir dados incorretos ou incoerentes na base de dados. Após uma exposição prolongada a dados incoerentes, torna-se difícil testar o sistema como um todo, sendo necessário recriar a base de dados. Utilizando a configuração SQLite no NHibernate, basta apagar o ficheiro da base de dados que o mesmo é recriado no próximo arranque do servidor. Contudo, ao realizar esta limpeza radical é necessário readicionar utilizadores, agentes, equipas, etc., processo monótono e aborrecido. O DummyController surgiu naturalmente como uma forma de adicionar dados de teste ao sistema. Basta fazer um pedido a [endereço_servidor]/api/dummy/team para criar uma equipa nova no sistema. Substituindo a parte final do endereço é possível criar outros objetos, tais como agentes, ocorrências, etc. A criação dos dados é efetuada com recurso a uma biblioteca implementada, denominada Tools.RandomData que possui funcionalidades para criar dados aleatórios de vários tipos, desde dados primitivos como números e datas até dados mais complexos como posições aleatórias de listas e enumerações, texto aleatório ou mesmo coordenadas aleatórias (sendo possível definir uma área, como por exemplo coordenadas portuguesas). Quando o servidor entrar em produção está prevista a desativação deste controlador.

Ao implementar os restantes controladores que se destinam a ser utilizados para suportar a implementação do *plugin desktop* do SAFER, rapidamente se chegou à conclusão de que estes tinham uma estrutura em comum, com os mesmos métodos e muita lógica repetida. Perante tal facto, implementou-se um controlador base para estes controladores, intitulado ControllerBase. Este controlador é genérico, adaptando-se a qualquer tipo de entidade do modelo de domínio, sendo apresentado na Figura 25 um excerto do seu código.

Um controlador é especializado num tipo de entidade, sendo implementado com recurso a *generics*. O código apresenta-se estruturado em seis secções: campos/propriedades, construtor e pedidos/métodos GET, POST, PUT e DELETE.

No que diz respeito a campos e propriedades, todos os controladores têm um repositório, pois todos eles irão, a certa altura, obter ou alterar dados. O tipo de repositório é definido consoante o controlador. Para além do repositório, o controlador define uma propriedade referente a uma mensagem de erro que se destina a ser utilizada pelos controladores que estendam ControllerBase. Por conveniência, é definida uma propriedade que indica (ou não) a ocorrência de um erro durante a operação atual. O construtor inicia o repositório com um objeto que tenha sido injetado via DI.

```
public class ControllerBase<T> : ControllerBaseUow where T : EntityBase
{
    public IBaseRepository<T, Guid> Repo { get; private set; }
    public string ErrorMessage { get; protected set; }
    protected bool IsErrorState { get { return ! ErrorMessage.IsBlank(); } }

    public ControllerBase(IBaseRepository<T, Guid> repository) { /* … */ }

    public virtual HttpResponseMessage Get(Guid id) { /* … */ }
    public virtual HttpResponseMessage Get() { /* … */ }

    public HttpResponseMessage Post(T entityObj) { /* … */ }
    protected virtual bool IsValidStatePost(T newObj) { /* … */ }
    protected virtual bool IsPostConflict(T newObj) { /* … */ }
    protected virtual void TransformPost(T newObj) { /* … */ }

    public HttpResponseMessage Put(T entityObj) { /* … */ }
    protected virtual bool IsValidStatePut(T updatedObj) { /* … */ }
    protected virtual bool IsPutConflict(T updatedObj) { /* … */ }
    protected virtual void TransformPut(T updatedObj) { /* … */ }

    public HttpResponseMessage Delete(Guid id) { /* … */ }
    protected virtual void TransformDelete(T objToDelete) { /* … */ }
}
```

Figura 25 - Controlador base.

Alguns controladores podem necessitar de mais do que um tipo de repositório. Para obter múltiplos repositórios, basta que definam um construtor que receba por parâmetro os repositórios desejados, que os mesmos serão injetados do mesmo modo. Os métodos destinados à obtenção de dados (Get*) são implementados de forma simples: o repositório obtém todos os objetos do tipo de entidade associado ao controlador ou procura um determinado objeto por ID. O mecanismo mantém-se igual para qualquer controlador que derive de ControllerBase, não sendo necessário alterar comportamento em subclasses. Basta estender este controlador base para ter implementada toda a lógica de acesso a dados.

Contudo, para operações de inserção, alteração ou remoção existe comportamento específico de controlador para controlador. Antes de inserir ou alterar um objeto é necessário validar que este se encontra num estado válido para ser inserido no sistema (campos obrigatórios preenchidos, dados válidos, etc.). A título de exemplo, para um objeto da entidade utilizador (User), o objeto só é persistido caso o nome de utilizador esteja preenchido e não contenha caracteres inválidos. Ao inserir um objeto, e também necessário verificar se, este não irá entrar em conflito com um objeto existente. Para o mesmo exemplo, não podem existir dois utilizadores com o mesmo nome de utilizador no sistema. Para permitir a execução de lógica específica para diferentes controladores sem prejuízo de lógica comum, foram definidos métodos de apoio identificados pela *keyword virtual*, de modo a serem implementados em controladores que herdem de ControllerBase. A Figura 26 ilustra o processo de inserção de

uma entidade no sistema (método Post):

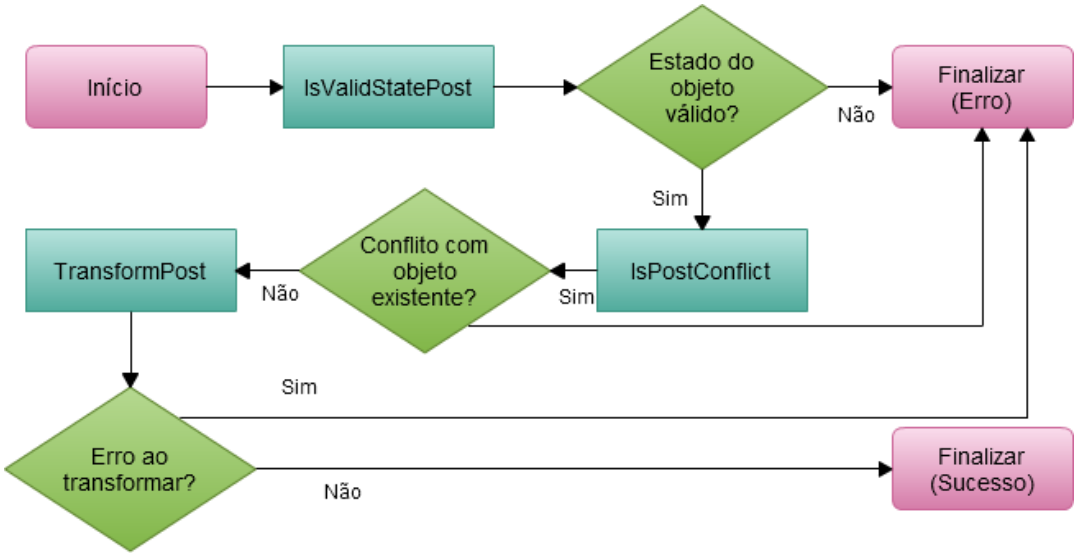


Figura 26 - Fluxograma da inserção de um objeto no sistema.

Antes de inserir o objeto, este é verificado consoante o seu estado e avaliado como conflituoso ou não. Caso passe estes dois testes, é aplicada uma transformação – o objeto pode vir incompleto, pode ser necessário o cruzamento com informação existente, alguns campos são preenchidos apenas pelo servidor, etc. Resultante desta transformação, em caso de sucesso, o pedido é finalizado sobre o objeto transformado. Este mecanismo permite que a mesma lógica – validar estado, validar conflitos, transformar e gravar – seja aplicável a qualquer controlador, enquanto que o comportamento específico – como são realizadas as validações ou transformações – é definido apenas em subclasses, sendo específico para cada controlador atendendo às necessidades individuais de cada entidade. A mesma estratégia é aplicada ao processo de alteração de um objeto existente, (método Put) mudando apenas os nomes dos métodos específicos.

Por fim, a operação de remoção de um objeto do sistema partilha apenas parte do mecanismo respeitante à transformação antes da remoção do objeto. Uma vez que se trata de uma operação para remover um objeto, não são necessárias as validações (excluindo validações de segurança transversais a todos os pedidos, tais como autenticação, etc.). Contudo, a transformação do objeto antes da sua remoção pode fazer sentido, uma vez que o objeto continuará a existir, não sendo removido mas marcado como tal e não aparecendo em consultas futuras.

Neste projeto foram implementados controladores para gestão de utilizadores (User), informação pessoal de indivíduos (PersonalInfo), agentes (Officer), equipas (Team), unidades de equipas (TeamUnit), agrupamentos de unidades (UnitGroup), equipamentos (Equipment), pontos de interesse geográficos (PointOfInterest), ocorrências (Occurrence), tipos de

ocorrência (OccurrenceType), missões (Mission) e subtipos de missões (MissionSubType).

4.4.8 Rastreamento de modificações

Para garantir o histórico, como já referido, os objetos no sistema não são removidos, apenas marcados como tal, não aparecendo em consultas futuras. Contudo, pode ser necessário obter o histórico de modificações para determinado objeto ou descobrir qual o utilizador responsável por determinada alteração. Por exemplo, pode ser relevante a obtenção do histórico de alterações para determinada ocorrência ou missão ou simplesmente necessário descobrir quem foi o utilizador que geriu determinada ocorrência no sistema. Para garantir esta funcionalidade, foi utilizada a biblioteca NHibernate.Envers [103]. Esta biblioteca é configurável, permitindo guardar o histórico de alterações para as entidades pretendidas, assim como o utilizador que realizou as alterações.

Para permitir validar de forma simples se uma entidade foi alterada, foi definida na classe EntityBase a propriedade LastModificationDate. Comparando este valor é possível determinar se a entidade se encontra atualizada ou não. Apesar de, com recurso à biblioteca NHibernate.Envers, ser possível determinar se uma entidade está alterada ou não, esta estratégia é aplicada também para suportar os clientes dos *webservices* do SAFER, que possuem assim uma forma simples de determinar se os dados foram atualizados desde o pedido anterior ou não. Assim, para as aplicações cliente é possível evitar, por exemplo, o *refreshing* desnecessário da User Interface (UI) – caso os dados não tenham sofrido alterações entre pedidos.

Esta propriedade é atualizada de forma automática no sistema com recurso ao padrão de desenvolvimento *Interceptor* [104], utilizando a interface *IInterceptor*, disponibilizada pelo NHibernate para o efeito. Implementando esta interface, é possível intercetar a inserção ou alteração de entidades no NHibernate, alterando a data de modificação no mesmo instante. Deste modo, é possível manter o histórico de modificações, assim como identificar um determinado objeto como estando atualizado ou não.

4.4.9 Segurança

A segurança dos dados das forças de segurança é assegurada através do uso de HyperText Transfer Protocol Secure (HTTPS) e da implementação de um mecanismo de autenticação e autorização de utilizadores baseado em perfis.

Ao utilizar HTTPS, é garantido que um pedido intercetado não pode ser lido de forma a obter informação útil, uma vez que os dados são encriptados durante a ligação [105].

No SAFER, o mecanismo de autenticação utilizado é Basic Authentication [106], [107], o qual envolve o envio de um nome de utilizador e de uma palavra-chave em cada pedido, de modo a poder autenticar o mesmo. Este mecanismo oferece facilidade e simplicidade de implementação, não sendo só por si seguro, uma vez que os dados circulam na rede de forma não encriptada. Este mecanismo só é seguro em conjunto com HTTPS, que garante a encriptação dos dados. Para suportar o mecanismo de autenticação e autorização dos utilizadores foi utilizado o ASP .NET Membership Provider [108]s para a gestão de utilizadores e perfis de acesso ao sistema. Note-se que um utilizador do *membership provider* não é o mesmo objeto que a entidade utilizador do modelo de domínio. A existência destes dois objetos (diferentes) justifica-se por razões de flexibilidade: caso se decida no futuro alterar o mecanismo de gestão de utilizadores, sem recorrer ao *membership provider*, a gestão de utilizadores do SAFER pode ser isolada sem afetar utilizadores existentes (no modelo de domínio). Existe no entanto uma associação entre estes dois objetos, garantida pelo controlador de utilizadores.

A autorização de acesso aos controladores é garantida pela utilização do *attribute Authorize* nos controladores, podendo ser especificado um perfil. Assim, diferentes perfis podem aceder a diferentes controladores/métodos dependendo do nível de privilégios associados ao perfil.

4.5 Aplicação *desktop - plugin*

No projeto SAFER, a aplicação *desktop* consiste num *plugin* que é adicionado à aplicação principal do sistema XTraN 4 Forces. Este *plugin* é responsável por gerir a lógica das Forças de Segurança, utilizando a aplicação servidora desenvolvida para o efeito como principal fonte de dados. O 4 Forces utiliza um sistema de *plugins* que permite carregar dinamicamente e incorporar aplicações variadas na aplicação principal, desde que as mesmas sejam desenvolvidas no formato de *plugin*. O 4 Forces quando é iniciado procura *plugins* na sua pasta, e caso encontre o *plugin* do SAFER, o mesmo é adicionado, passando a fazer parte integral da aplicação durante o ciclo de vida da mesma.

4.5.1 Arquitetura da aplicação *desktop*

A aplicação *desktop* tem uma arquitetura lógica multicamada orientando-se a serviços, tendo os *webservices* como principal fonte de dados. De um modo geral, é possível dividir a aplicação em três camadas diferentes: camada de dados, lógica de negócio e apresentação, de acordo com a Figura 27:

Figura 27 - Arquitetura da aplicação *desktop*.

A camada de apresentação é composta pelos formulários desenvolvidos para servir de ponto de entrada no sistema por parte dos utilizadores do SAFER. Esta camada comunica diretamente com a camada de lógica de negócio, que é responsável por decidir que informação é apresentada ao utilizador, assim como de que forma esta é apresentada e processada. Tratando-se de uma aplicação *desktop Windows Forms*, grande parte da lógica de negócio encontra-se presente nos *event handlers*, que respondem diretamente às ações do utilizador. Por último, surge a camada de dados. Esta camada é composta pela camada de acesso a dados e pelos dados propriamente ditos. O modelo de domínio é comum à aplicação servidora e à aplicação cliente, evitando a replicação de código. O acesso a dados é realizado por um utilitário que simplifica e abstrai o acesso aos mesmos, denominado WebserviceUtil.

4.5.2 Bibliotecas de controlos

No SAFER, à semelhança dos restantes *plugins* integrantes do 4 Forces, são utilizadas bibliotecas de controlos para estender os controlos disponibilizados pela plataforma .NET, adicionando novos controlos para que possam ser desenvolvidas aplicações que façam uso dos mesmos. Estes controlos visam, por um lado, providenciar um aspeto visual alternativo aos controlos fornecidos por omissão, e/ou por outro, estender as funcionalidades que os mesmos disponibilizam. Algumas destas bibliotecas são externas, enquanto que outras são internas à Tecmic, desenvolvidas para o 4 Forces ou ainda explicitamente para o SAFER neste estágio.

Krypton Toolkit [109] é uma biblioteca de controlos, desenvolvida pela empresa Component Factory [110], que disponibiliza de forma gratuita controlos com um aspeto visualmente apelativo, assim como comportamento adicional sobre alguns dos componentes *standard* .NET. A principal desvantagem desta biblioteca consiste na dificuldade em configurar certas propriedades dos controlos utilizando o editor do Visual Studio, nomeadamente a secção

"Design" das propriedades, que permite, entre outras ações, definir o *modifier* dos controlos (*private*, *protected*, *public*, etc.), sendo o valor atribuído por omissão *private*. Apesar de ser possível alterar o *modifier* através de código no ficheiro de *design* (.designer.cs), o mesmo é regenerado quando há alterações, causando por vezes instabilidades devido ao facto de se tratar de um ficheiro gerado automaticamente pelo editor. Devido a este problema, a utilização desta biblioteca é pontual no SAFER, apesar de ser utilizada em vários projetos e *plugins* do 4 Forces. Assim, esta biblioteca é utilizada apenas em formulários que se se julga não haver necessidade de virem a ser estendidos.

A biblioteca Qios DevSuite, à semelhança da Krypton Toolkit, disponibiliza controlos avançados, com funcionalidade adicional aos controlos *standard* e possibilidades de customização tanto a nível de comportamento como de aspeto. Ao contrário do que sucede com a biblioteca Krypton, a Qios não sofre do problema mencionado, sendo por isso fortemente utilizada tanto no 4 Forces em aplicações várias como no SAFER. Funcionalidades e comportamentos disponibilizados por estes controlos incluem estilos arredondados, gradientes, múltiplos estados personalizáveis (que cor deve assumir determinado botão quando o rato passa por cima, que ícone deve ser mostrado, etc.), suporte a controlos TextBox com campo "CueText" – texto cinzento que aparece quando o controlo não tem texto inserido – através do controlo QTextBox, menus com estilo Ribbon, etc. A utilização desta biblioteca no 4 Forces não é apenas encorajada na Tecmic, mas obrigatória (nos menus que se situam na barra superior do *form*). O motivo desta obrigatoriedade prende-se com o sistema de *plugins* utilizado, que faz uso desta biblioteca, como é detalhado mais à frente.

Existem outras bibliotecas, internas à Tecmic que permitem criar componentes que servem diversas necessidades, sendo encorajado, sempre que possível, o isolamento dos controlos reutilizáveis em projeto separado (no Visual Studio) de modo a poderem ser utilizados em múltiplos projetos. Da implementação do SAFER surgiram vários controlos interessantes que foram movidos para um projeto próprio, intitulado "ReusableControls". Mais informação sobre esta biblioteca pode ser consultada no Anexo IV.

4.5.3 Tools.Utils

No decorrer da implementação do projeto SAFER surgiram tipos de dados e métodos que se enquadram naturalmente na categoria de utilidades. Estes métodos/tipos de dados, reutilizáveis em vários projetos, foram transferidos para um projeto próprio, denominado Tools.Utils.

Os métodos implementados servem o propósito de suporte e de atalho para funcionalidades que utilizem os tipos de dados da plataforma .NET. Os métodos deste projeto dividem-se em dois *namepsaces* principais, de acordo com a finalidade dos mesmos: Tools.Utils e Tools.Extensions. Já os tipos de dados novos enquadram-se sempre no *namespace* Tools.Utils. Utilitários mais específicos viriam a dar origem a dois subprojectos (na forma de projetos isolados, apesar de seguirem a mesma lógica de *namespaces*) – o projeto Tools.Utils.Control e o projeto Tools.Utils.RestSharp.

O propósito do projeto Tools.Utils.Control é disponibilizar *extension methods* para controlos (exclusivo para projetos que lidem com interfaces gráficas *WinForms*), enquanto que o Tools.Utils.RestSharp encapsula um utilitário, denominado RestSharp [111] que permite realizar pedidos REST, sendo utilizado no SAFER para comunicar com os *webservices* desenvolvidos. Apesar de serem utilitários reutilizáveis, encontram-se em projetos separados por motivos de separação de responsabilidades, assim como para manter os vários projetos livres de referências desnecessárias. A título de exemplo, os *webservices* fazem uso do Tools.Utils, mas não lidam com controlos *WinForms* nem lidam com pedidos REST (pelo menos na ótica do consumidor). Assim, não faria sentido o projeto dos *webservices* ter de referenciar o *namespace* System.Windows.Forms, que fica, desta forma, apenas obrigatório para os projetos que lidem efetivamente com *WinForms*. A mesma lógica é aplicável para o projeto Tools.Utils.RestSharp.

O projeto Tools.Utils é referenciado e utilizado por todos os projetos implementados, assim como por outros projetos do 4 Forces não diretamente relacionados com o SAFER.

4.5.4 Sistema de *plugins*

O sistema de *plugins* é uma das grandes vantagens do sistema XTraN 4 Forces. Esta abordagem permite incorporar várias aplicações numa só UI. Estas aplicações podem ser bastante distintas em modo de funcionamento e funcionalidades oferecidas, sendo que o único requisito para que uma aplicação possa ser considerada *plugin* é a conformidade com a estrutura de *plugins* utilizada.

Esta arquitetura de *plugins* assenta em duas bibliotecas de controlos: Qios DevSuite e DockPanel Suite [112]. A biblioteca Qios é responsável por tratar da secção dos menus (*ribbon*), que são o ponto de entrada dos *plugins* para o utilizador. Tipicamente, ao serem utilizados, estes menus despoletam a abertura de formulários que são inseridos – *docked* – num DockPanel. Vários formulários podem coexistir em simultâneo no mesmo DockPanel, situação na qual irão ser representados automaticamente na forma de *tabs*. A Figura xx

apresenta uma imagem do GenericMainForm, o formulário que representa uma "janela principal", capaz de receber *plugins*. A janela principal do 4 Forces – MainForm – estende precisamente este mesmo formulário.

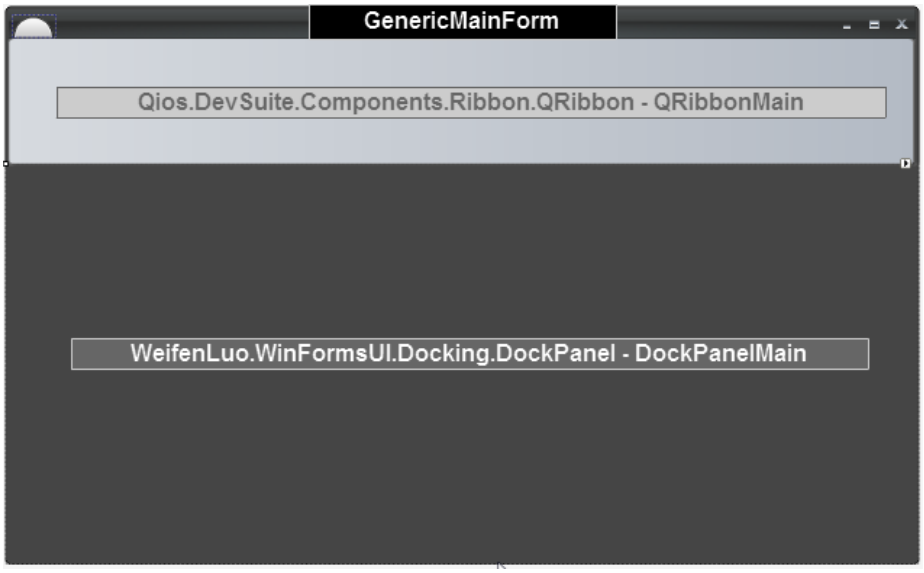


Figura 28 - GenericMainForm.

Para a criação de um *plugin* é necessária a criação de um projeto que contenha um formulário que estenda QRibbonForm e implemente a interface IPlugInModule. Esta interface define métodos para poder carregar o *plugin* (módulo), entre outros métodos de auxílio à estrutura de *plugins*. Neste formulário é definido um componente QRibbon, à semelhança do componente QRibbon do GenericMainForm. A este componente são tipicamente adicionados controlos QRibbonPage como forma de organizar os menus em secções de opções, contendo *menu items* relacionados uns com os outros. O carregamento dos *plugins* ocorre quando a aplicação é iniciada, e é da responsabilidade do GenericMainForm. Este começa por percorrer a pasta onde o mesmo se encontra à procura de ficheiros Dynamic-link Library (DLL). Para reduzir o número de *assemblies* de ficheiros DLL a inspecionar, apenas são considerados os *assemblies* cujo nome contenha "Plugins.Modules". Caso estes DLLs contenham tipos de dados que implementem a interface IPluginModule os mesmos são tratados, sendo carregados como *plugins* da aplicação. Quaisquer controlos QRibbonPage de *plugins* são adicionados ao componente QRibbon do GenericMainForm. Cada controlo QRibbonPage contém controlos QCompositeButton, ou equivalentes (Qcomposite*), que servem para realizar ações (que são os menus propriamente ditos). Geralmente, estes menus invocam ações que envolvem mostrar formulários. A única restrição que é apresentada para que estes sejam incorporados no DockPanel do GenericMainForm é a de estender DockContent (DockContent refere o tipo de *forms* que podem ser acoplados num DockPanel). A seguinte ilustra este enquadramento:

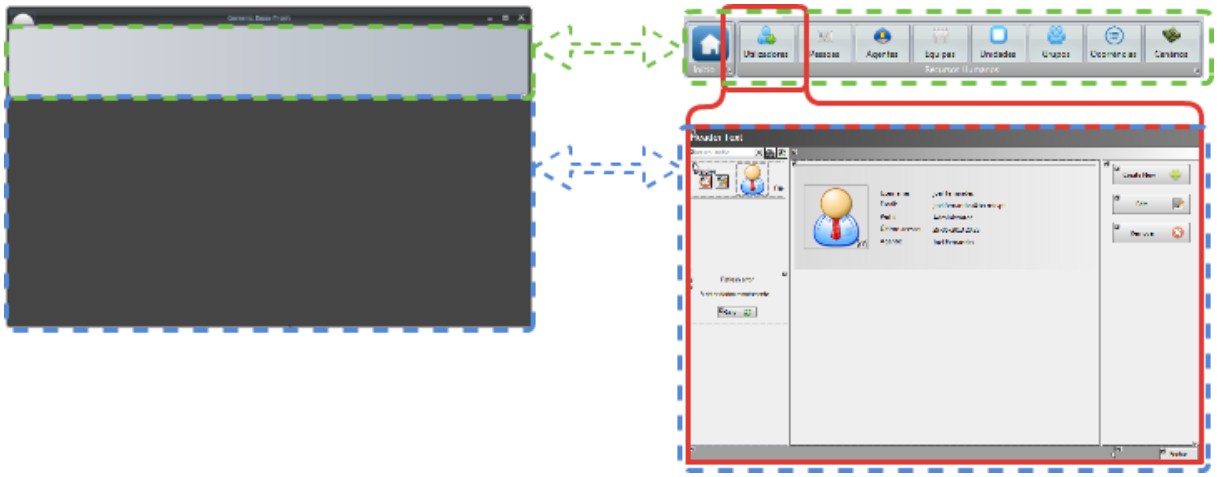


Figura 29 - Enquadramento de um *plugin* no GenericMainForm.

Quando uma janela que estenda DockContent é adicionada ao DockPanel, a mesma é representada como uma *tab* no GenericMainForm. Deste modo, múltiplas janelas podem coexistir no mesmo DockPanel.

Ao desenvolver o SAFER, foi criado um formulário que estende DockContent, denominado PluginForm, que se encontra na base de todas as janelas do projeto SAFER.

4.5.5 Plugin Forms

No decorrer da implementação do SAFER, surgiu a necessidade de criar uma janela base para as janelas a implementar no projeto. Neste contexto, criou-se o PluginForm – uma janela que serve para ser usada por *plugins* – que estende DockContent (que por sua vez herda de System.Windows.Forms). Inicialmente, esta janela foi pensada para definir apenas um cabeçalho da janela, assim como um rodapé que apresenta os botões Ok e Cancel. De modo a manter o aspeto da UI personalizável, é possível definir se são mostrados ambos os botões, apenas um ou mesmo nenhum. De igual modo, o cabeçalho e o rodapé da janela podem ser escondidos por opção própria. A restante área é ocupada por um painel que compõe a área "útil", que diverge entre diferentes subclasses. Na figura seguinte é apresentada a janela Pluginform:

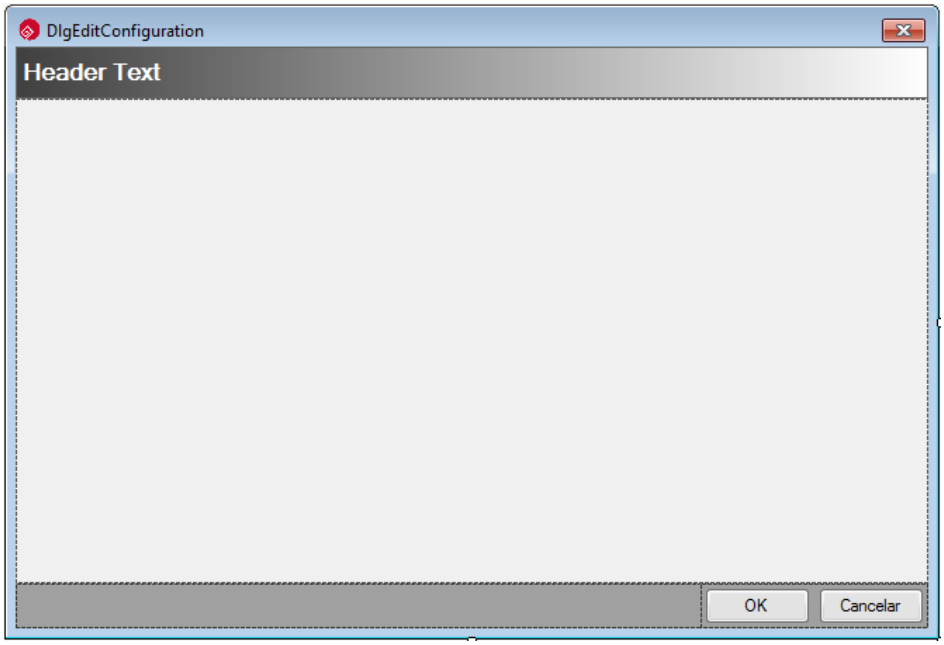


Figura 30 – Janela PluginForm.

Mais tarde esta janela acabou por evoluir, adicionando capacidade de deteção de alterações realizadas pelo utilizador (adicionando o caracter "*" quando o utilizador insere texto num *TextField* ou seleciona uma *CheckBox*, por exemplo). O PluginForm implementa ainda a interface IFormValidator que disponibiliza métodos que permitem validar os dados da janela utilizando a biblioteca Tools.ControlValidation. O funcionamento desta biblioteca pode ser consultado no Anexo V. Por se tratar de uma janela facilmente reutilizável em vários *plugins*, a classe que compõe esta janela foi movida para o projeto ReusableControls.

Partindo do PluginForm, procedeu-se à criação da primeira janela do SAFER: a janela de gestão de utilizadores. Esta janela serviu como caso piloto, definindo várias funcionalidades: visualização de uma lista de utilizadores existentes; criação, edição, pesquisa e visualização detalhada de utilizadores; visualização de informação de um modo diferente caso se esteja a criar novo agente, editar ou visualizar o mesmo. Naturalmente, ao criar uma nova janela de gestão – desta vez para gerir informação de pessoas no sistema – chegou-se à conclusão de que a mesma iria disponibilizar o mesmo conjunto de funcionalidades de visualização, gestão e pesquisa. Com este aspeto em mente surgiu a janela EntityManager. Esta janela herda de PluginForm e encontra-se ilustrada na figura seguinte:

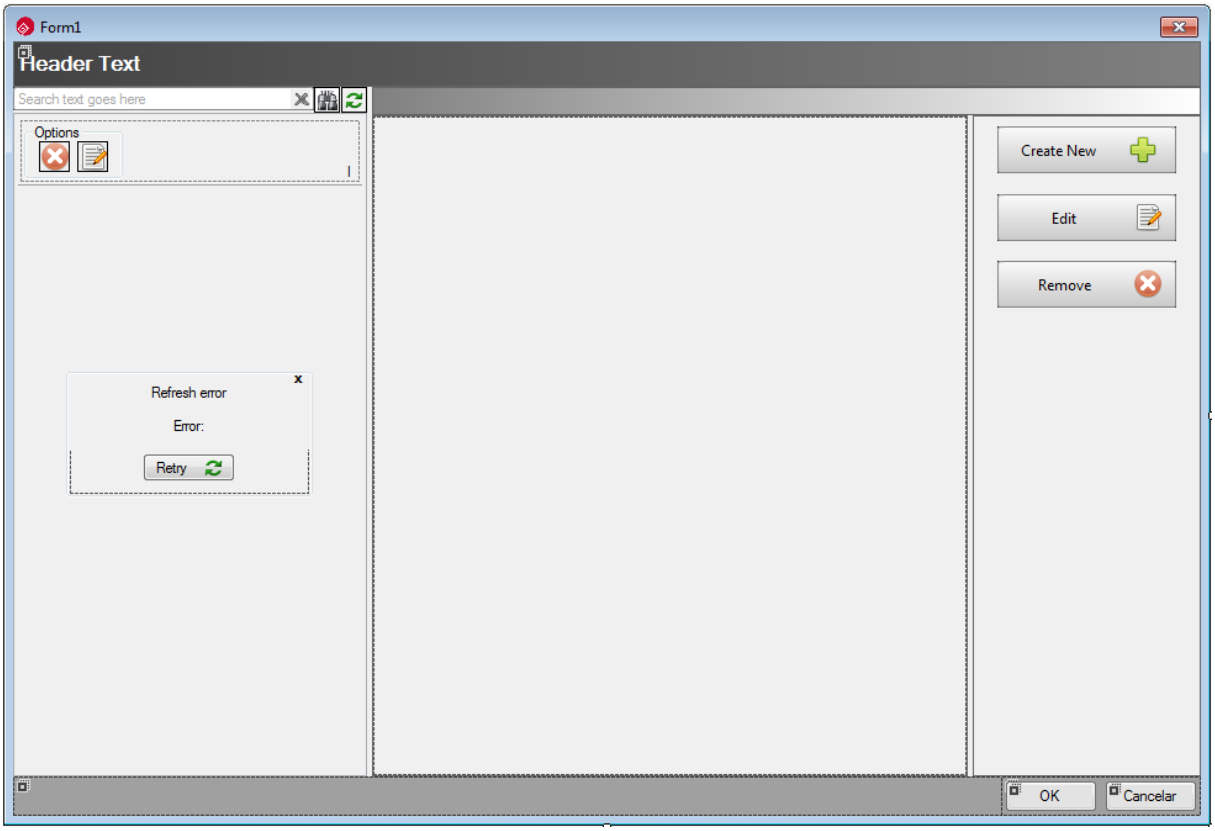


Figura 31 - Janela EntityManager.

Em relação à janela PluginForm, a janela EntityManager apresenta notórias diferenças a nível visual. Em primeiro lugar, nota-se a divisão clara da área de trabalho em três secções distintas: área de lista de entidades, área de visualização/edição de informação e área de menus. Do lado esquerdo existe uma lista, destinada a apresentar as entidades existentes, podendo estas ser filtradas através de uma barra de pesquisa. Ao lado da barra de pesquisa existe um botão que serve o propósito de permitir a atualização dos dados da lista. Esta janela não prevê nenhuma fonte de dados em particular, mas admite que a obtenção de dados possa eventualmente demorar vários segundos (v.g. ler um ficheiro grande, ligar-se a uma base de dados remota, ligar-se a *webservices*, etc.). Deste modo, foi definido um *popup* que permite dar *feedback* ao utilizador durante a atualização de dados ou quando existe uma falha de atualização dos mesmos. Na figura não estão apresentados itens, mas apenas um *template* base para um item. Cada item pode ser visualizado em detalhe, editado ou removido, sendo definidos botões no *template* para o efeito (é possível configurar quais destes botões são efetivamente mostrados). Do lado direito são mostrados botões que permitem acesso rápido às funcionalidades de criação de novos objetos ou edição/remoção do objeto que se encontre a ser visualizado. Finalmente, a área do meio é ocupada por três painéis sobrepostos que se destinam a mostrar a informação ao utilizador de modos diferentes consoante a operação que o utilizador está a realizar sobre as entidades: visualizar, editar ou criar um objeto. Este

conceito permite que diferentes vistas e operações sobre os dados possam existir para diferentes modos de operação – conceito de UiMode: None; Create; Edit; View - e tem como vantagem principal a distinção efetiva entre os controlos a apresentar ao utilizador, assim como o seu aspeto ou operações que permite. A título de exemplo, ao ser criado um utilizador pode definir-se o seu *username*, o qual não pode ser alterado em modo de edição. Através do conceito de UiMode é possível apresentar a informação ao utilizador de forma diferente para cada modo. A vantagem principal também constitui a principal desvantagem desta abordagem: aumenta-se significativamente a quantidade de código a escrever, assim como o número de controlos a criar.

A janela EntityManager foi pensada para trabalhar com um tipo de entidade (apesar de não saber que tipo de entidade se trata, ou quais as suas origens). Para se implementar este comportamento, pensou-se inicialmente em tornar esta classe num formulário genérico – EntityManager<T>. Infelizmente, o Visual Studio (VS) não é capaz de desenhar um formulário genérico no seu *designer*. Para não perder o suporte ao *designer* do VS foi necessário recorrer a um truque: criar uma subclasse que não seja genérica e colocar a versão não genérica em primeiro lugar no ficheiro. Deste modo, o VS permite desenhar a versão não genérica, mas obtêm-se o comportamento desejado. Assim, no mesmo ficheiro a janela EntityManager deu origem a uma subclasse denominada ManagerGeneric (versão genérica). Este expediente resulta em pares de classes (versão não genérica/versão genérica) que representam o mesmo formulário. Na prática, falar da classe EntityManager ou da classe ManagerGeneric é indiferente. Este expediente resulta em mais código mas evita a perda de suporte do *designer* no VS.

Como foi já referido, a janela EntityManager é genérica em termos de tipo de entidade a apresentar ou fonte de dados. Por este mesmo motivo, estas classes encontram-se no projeto ReusableControls, em conjunto com a sua base PluginForm.

No decorrer do desenvolvimento surgiu a necessidade de implementar a versão "SAFER" da janela EntityManager – uma janela genérica especializada num tipo de entidade específica do modelo de domínio do SAFER – XTraN4ForcesFSDomain. Neste contexto, surgiu a janela EntityManagerGeneric. Ao contrário da sua superclasse, esta classe já faz parte do *plugin*, não tendo sido pensada para ser reutilizada fora do *plugin* do SAFER. A nível visual, este novo nível hierárquico não apresenta alterações em relação ao anterior. As principais vantagens que o EntityManagerGeneric oferece relacionam-se com o conhecimento do tipo de entidade, origem dos dados (tipo de *datasource*) e ficheiro de *Resources* do SAFER. Ao ter conhecimento de que os dados provêm de *webservices* esta camada permite escrever lógica de

obtenção de dados capaz de obter os mesmos de uma forma genérica. Do mesmo modo, ao possuir conhecimento de que os dados são entidades do modelo de domínio (XTraN4ForcesFSDomain), é possível listar os mesmos de uma forma mais eficaz: todas as entidades possuem uma data referente à última modificação que permite que a lista na janela só seja atualizada se existirem dados novos em pedidos futuros aos *webservices*. Finalmente, o facto de conhecer o ficheiro de *Resources* do *plugin* implica a possibilidade de mostrar mensagens ao utilizador carregadas a partir do mesmo ficheiro, obtendo-se os benefícios associados.

A estrutura de formulários apresentada permite poupar uma quantidade considerável de trabalho ao implementar lógica comum aos vários formulários que gerem informação das entidades do sistema, uma vez que um conjunto de funcionalidades comuns se encontra já semi-implementado para que as janelas finais não tenham de repetir uma quantidade de código bastante considerável. As funcionalidades que a janela EntityManagerGeneric disponibiliza são enumeradas de seguida:

- Botões de Ok/Cancel: suporte a botões Ok e/ou Cancel configuráveis, assim como lógica pré-definida para as ações dos mesmos. A título de exemplo, quando o botão Ok é pressionado, caso a janela esteja em modo de criar novo objeto, os dados são validados automaticamente e, caso sejam válidos, o objeto é enviado para os *webservices* para ser persistido.
- *Tracking* de alterações: esta funcionalidade desenvolvida para o SAFER permite detetar *input* por parte do utilizador em determinados componentes e colocar a janela num estado *Dirty*, representado visualmente por um asterisco (*) à frente do nome na *tab* correspondente. Quando a janela se encontra neste estado é apresentada uma mensagem de confirmação (do descarte de alterações realizadas) ao utilizador caso este feche a janela ou tente visualizar objetos diferentes. Para adicionar um controlo a este mecanismo faz-se *override* do método TrackChanges, e regista-se o controlo pretendido.
- *String Resources*: o texto dos conteúdos e mensagens apresentadas ao utilizador encontra-se definido em ficheiro próprio e não *hardcoded* no projeto. Deste modo, é possível escalar a aplicação mais tarde se necessário, de modo a facilitar a internacionalização.
- Validação de dados preenchidos: através da biblioteca de validação de controlos criada no decorrer deste projeto – denominada Tools.ControlValidation (mais informação disponível no Anexo V) – é possível a validação personalizada dos controlos da

janela, antes de criar um novo objeto ou atualizar um objeto existente.

- Lista de objetos: suporte a uma lista visual de objetos com um *template* personalizável. Por omissão, o *template* base é composto por botões de edição/remoção, assim como uma *label* identificadora (escondida) contendo o Id do objeto a ser mostrado. Para editar o conteúdo de cada item na lista basta adicionar controlos ao mesmo e definir o *data-binding* entre os controlos e as entidades. Este processo é internamente facilitado, sendo apenas necessário fazer *override* ao método BindDataRepeaterToList e realizar as operações de *data-binding* específicas.
- Pesquisa de objetos: através de uma barra de pesquisa é possível filtrar os elementos apresentados na lista. Os critérios de pesquisa variam de entidade para entidade e é possível a customização dos critérios de pesquisa através do *override* de uma *property* denominada SearchBy, como ilustrado na figura seguinte:

```
protected override Expression<Func<User, object>> SearchBy
{
    get { return (x) => new { x.Username, x.Role, x.Email }; }
}
```

Figura 32 - Exemplo de *overriding* para pesquisa customizada.

No exemplo anterior (retirado da janela de gestão de utilizadores) é possível a pesquisa de utilizadores através das propriedades nome de utilizador, perfil ou *email*. Internamente a pesquisa é realizada com recurso ao método ToString das propriedades selecionadas. Este mecanismo é adaptável a qualquer tipo de entidade.

- Atualização de dados: Os dados são atualizados automaticamente com base num período configurável. Caso se deseje uma atualização menos frequente dos dados (v.g. cinco em cinco minutos) mas seja pretendida uma atualização imediata entre pedidos, existe um botão que permite forçar a atualização dos dados, sobrepondo-se ao mecanismo de atualização automático.
- Modos de UI: o mecanismo de UiMode permite, para a mesma janela, apresentar diferentes conteúdos ao utilizador em diferentes situações, caso este se encontre a criar um novo objeto, ou a visualizar/editar um objeto existente, assim como disponibilizar diferentes ações.
- População da UI com dados**:** a UI é populada (dados inseridos nos controlos) de um modo uniforme, desde que as janelas (subclasses) façam *override* do método FillInInfo e populem a UI neste método. A não conformação com este sistema pode induzir a janela num estado *Dirty* de forma errónea. Ao realizar *override* do referido

método garante que o preenchimento da UI com dados é realizado de forma consistente e sem problemas.

- Botões de criação, edição e remoção**:** estes encontram-se disponíveis de um modo automático consoante o UiMode (criação, edição, visualização ou nenhum) em que a janela se encontre num determinado momento. A comutação entre diferentes UiModes é também realizada de modo automático.
- Obtenção e persistência agnóstica**:** os dados são obtidos uniformemente do mesmo modo por qualquer janela, não lidando diretamente com a camada de acesso aos *webservices*, responsabilidade delegada para um componente denominado WebserviceUtil – explicado posteriormente neste documento – simplificando o processo de CRUD dos dados.

As janelas que permitem gerir os dados da aplicação foram implementadas sobre a EntityManagerGeneric. Neste grupo de formulários de gestão – tipo Manager – foram desenvolvidas janelas que permitem gerir utilizadores, agentes, informação sobre pessoas, equipas, unidades, grupos, ocorrências, missões e equipamentos.

No decorrer do desenvolvimento, chegou-se à conclusão que em vários casos era necessário que o utilizador pudesse selecionar objetos de entidades, que não as geridas pelo formulário onde se encontrava. A título de exemplo, uma equipa possui um conjunto de agentes. Deste modo, para atribuir agentes à equipa foi pensada a criação de uma lista de agentes no formulário de gestão de equipas. Contudo, esta abordagem envolve replicação de código no caso de vários formulários permitirem ao utilizador a seleção de agentes. Neste contexto, surgiu o conceito de Chooser. Um Chooser é um formulário que se destina à listagem de entidades que permita ao utilizador selecionar um ou mais objetos das mesmas entidades. À semelhança do EntityManager, foi desenvolvido um EntityChooser (e a sua classe genérica ChooserGeneric<T>), pelas mesmas razões que justificaram a criação da classe ManagerGeneric, já apresentadas.. Este formulário define vários controlos: uma barra de pesquisa que permite filtrar os resultados da lista; um botão que permite atualizar os dados; um *popup* que permite dar *feedback* ao utilizador durante a atualização ou em caso de erro; a lista que apresenta os objetos dos quais o utilizador pode selecionar; botões de *Ok* e *Cancel*.

Este controlo adapta parte da lógica do formulário EntityManager. À sua semelhança, a barra de pesquisa, o botão de atualização e o *popup* de progresso funcionam do mesmo modo. Também a lista define um *template* que inclui uma *label* identificadora (escondida), um controlo CheckBox e um controlo RadioButton. Estes dois controlos nunca estão visíveis ao

mesmo tempo. O formulário EntityChooser permite dois modos de seleção – *Single* e *Mutiple*. Tal como os nomes indicam, o modo de seleção *Single* permite apenas que o utilizador selecione um objeto da lista, sendo apresentado o controlo RadioButton em cada item, o que dá ao utilizador a pista visual de que só pode escolher um dos itens apresentados. Neste modo um duplo clique sobre o item irá automaticamente selecionar o mesmo e terminar o *input* (o formulário será fechado, sendo o resultado devolvido à janela que chamou o Chooser). O modo de seleção *Multiple* permite a seleção de múltiplos objetos de uma só vez. Cada item é apresentado com uma CheckBox, indicando precisamente que múltiplos objetos podem ser escolhidos simultaneamente.

Para além da barra de pesquisa, é possível ainda adicionar uma função que realize uma pré-filtragem, excluindo alguns elementos que satisfaçam determinada condição (v.g. excluir da seleção de agentes objetos que já se encontrem atribuídos a equipas, etc.). A figura seguinte ilustra este mesmo formulário:

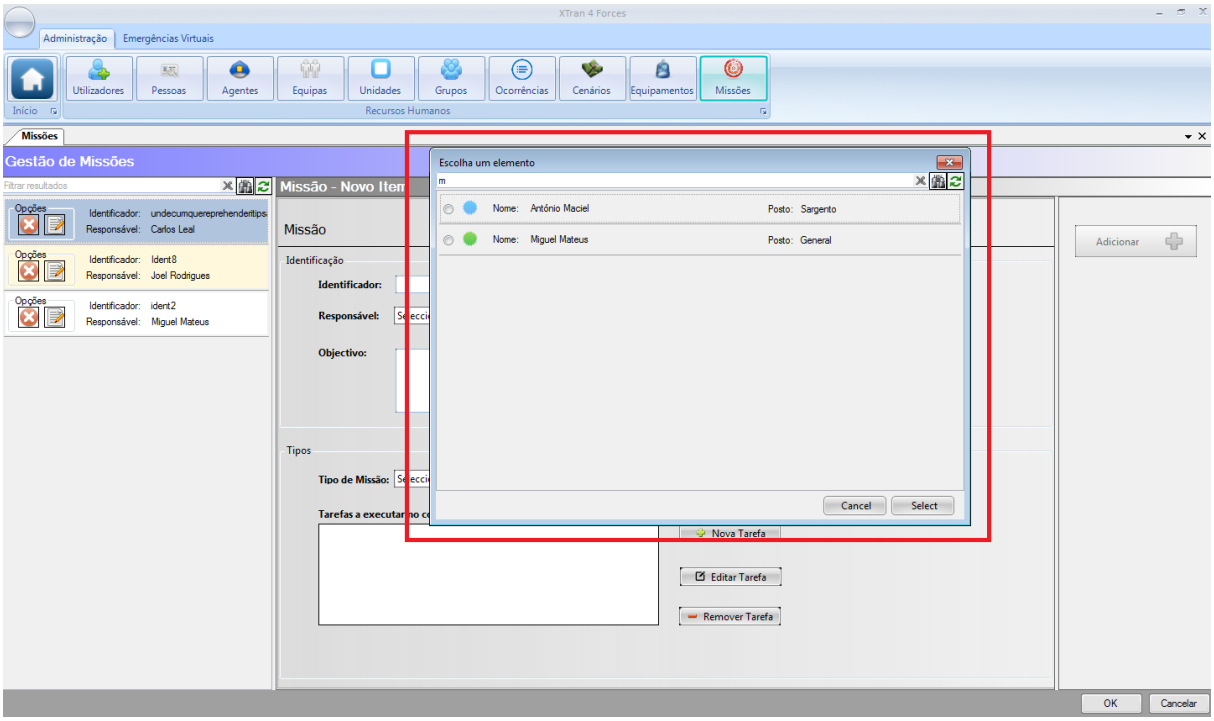


Figura 33 - Formulário EntityChooser.

Este controlo não tem conhecimento do tipo de entidade que irá apresentar ou a fonte de dados utilizada, sendo por isso adicionado ao conjunto de controlos reaproveitáveis ReusableControls. Seguindo a mesma linha de raciocínio que nos formulários de gestão, foi implementada a versão "SAFER" deste formulário: EntityChooserGeneric. Os formulários do tipo Chooser são lançados em modo *Modal*, bloqueando a UI até que tenha sido selecionado um objeto (somente nesta altura o botão OK estará disponível para ser clicado) ou o utilizador cancele a operação. Estes formulários encapsulam uma parte significativa da lógica de

apresentação e interação com o utilizador, e quaisquer subclasses apenas têm de adicionar controlos ao *template* do item a ser listado e finalizar o processo de *data-binding*. Para além da facilidade em criar novos formulários do tipo Chooser, a principal vantagem desta abordagem consiste na reutilização dos mesmos.

Ao implementar os formulários Chooser, verificou-se que algumas operações exigem que seja possível não só escolher mas também criar novo objetos quando necessário. Foi assim que surgiram os formulários de criação – Creator. Estes funcionam de modo *Modal*, à semelhança dos Choosers, servindo no entanto para mostrar controlos que permitem criar um novo objeto. O par de classes que implementa este tipo de formulários é EntityCreator/CreatorGeneric, e a versão "SAFER" denomina-se EntityCreatorGeneric. A nível visual este formulário não difere da sua base PluginForm (descrita mais aciona nesta secção). Contudo, apesar de visualmente não divergirem, estes formulários encapsulam toda a lógica necessária que facilita a implementação de um formulário que permita criar novos objetos: validação, inicialização de *strings* e persistência agnóstica (apenas na versão SAFER). Esta lógica é semelhante à utilizada pelos formulários do tipo Manager.

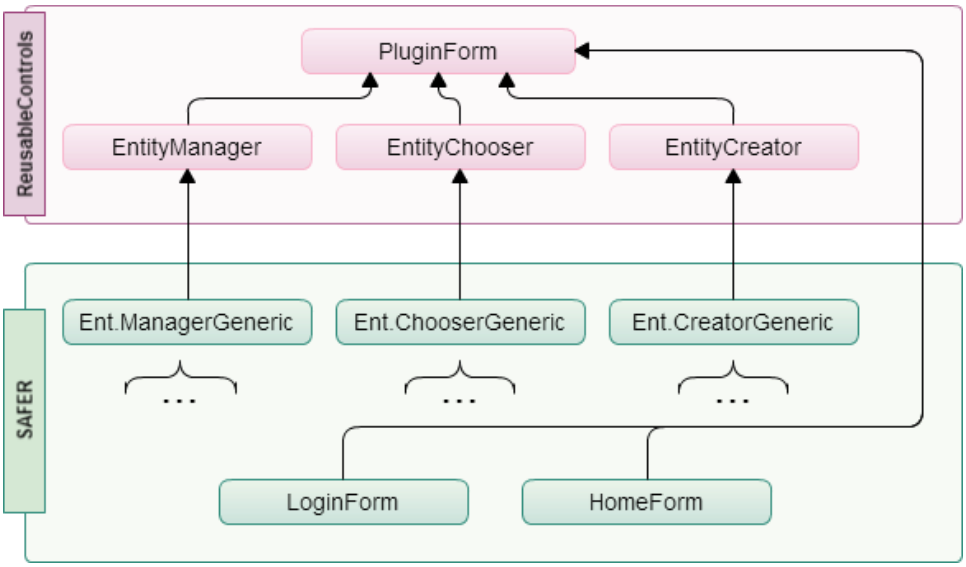


Figura 34 - Hierarquia de formulários no SAFER.

Em suma, e tal como ilustra a figura acima, a janela PluginForm é a base de todas as janelas do *plugin* SAFER. Deste *plugin* derivam três tipos de janelas: Manager, Creator e Chooser. Estas versões reutilizáveis originam as versões específicas para o SAFER, e dessas versões derivam as restantes janelas, consoante a funcionalidade pretendida. As janelas finais na figura encontram-se representadas pelas reticências. Neste trabalho foram implementadas janelas do tipo Manager para variadas entidades. Os nomes das janelas do tipo Manager obedecem à regra [nome da entidade]ManagerForm. De um modo semelhante, os formulários do tipo Chooser obedecem à regra [nome da entidade]Chooser e o mesmo sucede para os

formulários do tipo Creator. Estes nomes não são obrigatórios, mas são recomendados, por motivos de legibilidade. Em adição, o projeto no Visual Studio contém pastas para os formulários do tipo Manager, Chooser e Creator. Esta estrutura associada a nomes coerentes facilitam o trabalho por parte do programador para navegar no projeto e encontrar os formulários pretendidos. Do tipo Manager implementaram-se formulários para utilizadores, agentes, equipas, unidades, grupos, informação pessoal, equipamentos, cenários, ocorrências e missões. Para o tipo Chooser criaram-se formulários para agentes, equipas, unidades, grupos, utilizadores, informação pessoal, cenários e ocorrências. Para o tipo Creator criaram-se formulários para informação pessoal, grupos e equipas.

Existem ainda janelas que, por não se enquadrarem em nenhuma destas categorias, herdam diretamente de PluginForm.

4.5.5 Gestão de formulários

Ao desenvolver o *plugin* SAFER surgiu a necessidade de gerir os formulários que se encontram ativos em determinado momento. Por um lado, esta gestão permite atribuir aos menus uma cor caso a janela correspondente esteja ativa/aberta e, por outro lado, evita que surjam janelas duplicadas na aplicação. Uma vez que esta gestão pode interessar transversalmente a vários *plugins*, criou-se o projeto PluginFormManagement. Este projeto funciona como uma *cache*, contendo uma lista de formulários que se encontram abertos na aplicação. Ao ser clicado um item do menu para determinado *plugin*, tipicamente é criada e/ou aberta uma janela. Com este projeto, a classe PluginFormManager – responsável por gerir a cache com os formulários carregados – verifica se o tipo de janela já se encontra na *cache* ou não, criando o formulário caso não exista e ativando o mesmo (selecionar a *tab* correspondente) em caso contrário.

Quando um formulário é ativado através da utilização do gestor de formulários é guardada uma referência para o controlo (menu) que o invocou, sendo alterada a cor do controlo de modo a refletir que a janela se encontra ativa. Quando o formulário é fechado a cor original do controlo do menu é reposta, tal como ilustra a figura seguinte:

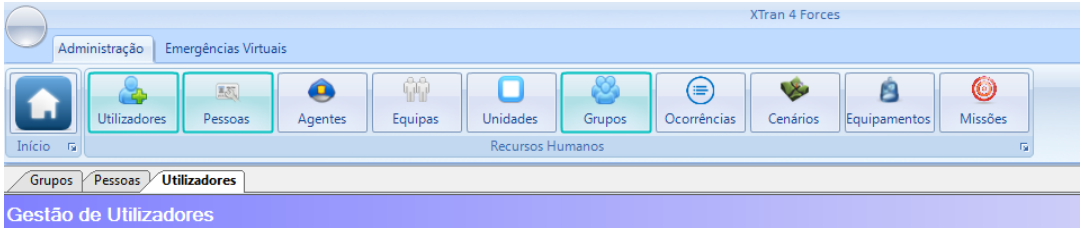


Figura 35 - Mudança de cor em menus utilizando PluginFormManager.

Para facilitar o carregamento de formulários criou-se um *extension method* para a interface

IPlugInModule que permite carregar o formulário de modo automático, cujo código é ilustrado na figura seguinte:

```
public static T LoadPluginForm<T>(this IPlugInModule plugin,
    DockPanel dockPanel, object menu = null)
where T : PluginForm, new()
{
    T frm = PluginFormManager.TryGetForm<T>();
    if (frm == null)
    {
        frm = new T();
        frm.Plugin = plugin;
        frm.Show(dockPanel);
        frm.DockState = DockState.Document;
        PluginFormManager.LoadForm(frm,menu);
    }
    else
    {
        frm.ExecuteWInvokeCheck(f=>f.Show(dockPanel));
    }
    return frm;
}
```

Figura 36 - Método de carregamento de um formulário no gestor.

Este método recebe como parâmetro genérico o tipo de formulário a criar. Os parâmetros normais consistem no objeto DockPanel (que consiste no objeto ao qual o formulário criado é acoplado e que provém do formulário principal) e, opcionalmente, no controlo de menu cujo *event handler* chamou este método. Este último parâmetro encontra-se definido como *object* e não como *Control* porque os controlos QCompositeButton não estendem de *Control*, como seria de prever[1]. Todavia, o projeto valida-o internamente e, caso seja do tipo Button ou QCompositeButton, as cores são modificadas automaticamente consoante o formulário esteja carregado ou não.

O fluxo de informação consiste na obtenção de um formulário da *cache* (ou *null* caso não se encontre carregado). Caso seja devolvido *null*, o formulário é criado utilizando *Reflection* ou então é mostrado, caso já exista na *cache*. Deste modo, garante-se que existe apenas um formulário do mesmo tipo em determinado momento.

6.5.6 Camada de acesso a dados

A camada de acesso a dados (DAL) é o ponto de acesso aos *webservices* desenvolvidos. Esta camada centraliza o acesso num único ponto facilitando não só a obtenção e persistência de dados como também o processo de *debugging* nas ligações. Esta camada permite a

[1] Os controlos cujo nome começa por "Q", como é o caso do QCompositeButton, são contolos importados da biblioteca Qios DevSuite, externa à Tecmic.

atualização periódica automática podendo, no entanto, ser utilizada para pedidos imediatos. Os pedidos imediatos podem surgir quando o utilizador ordena uma atualização imediata de uma lista de entidades (nos formulários Manager ou Chooser) ou quando os dados são persistidos/removidos. A classe que compõe a DAL denomina-se WebserviceUtil. A organização desta classe encontra-se seccionada na figura seguinte:

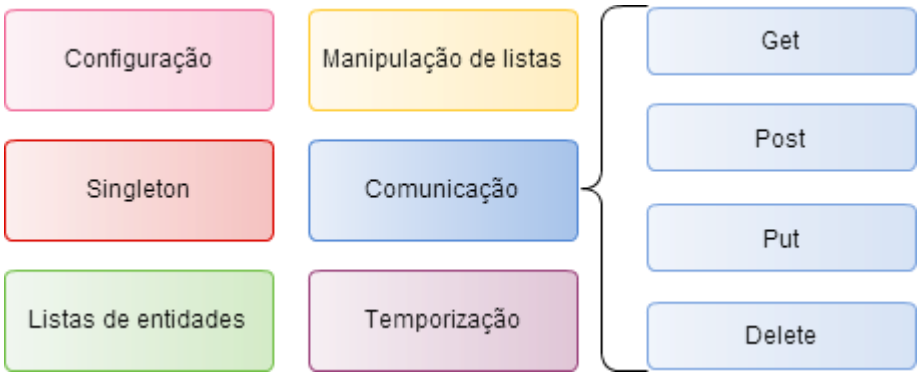


Figura 37 - Organização do WebserviceUtil

A secção de configuração contém propriedades e métodos que permitem configurar e obter configurações para parâmetros importantes tais como o endereço base para os pedidos ou o tempo que deve esperar entre pedidos de atualização automáticos. Estes valores são obtidos a partir de um ficheiro de configuração.

Segue-se a secção que contém a implementação do padrão de desenvolvimento *Singleton*. Para evitar que várias instâncias desta classe realizem pedidos em simultâneo, optou-se por tornar esta classe num *Singleton*.

A secção seguinte contém as listas de entidades. Para cada entidade que se pretenda gerir são necessárias duas propriedades: nome do controlador e lista de elementos. A título de exemplo, para a entidade do modelo de domínio User é necessário que se defina uma propriedade EntityController**User** com o valor "users" (uma vez que o controlador do *webservice* se chama UsersController) e uma propiedade List**User**. Nos nomes das propriedades destaca-se a negrito a palavra "User", que corresponde ao nome da entidade. Ambas as propriedades devem obedecer a esta norma aquando da criação das mesmas para qualquer entidade: EntityController[nome da entidade] e List[nome da entidade]. Esta abordagem permite aplicar técnicas de *Reflection* de um modo mais simplificado para operar com estas propriedades.

O processo de *Reflection* é definido na secção de manipulação de listas. Esta secção contém métodos que permitem instanciar e obter uma lista com base num parâmetro genérico que contenha apenas o tipo de entidade cuja lista se pretende criar/obter. Também se encontram definidos métodos que permitem manipular (operações CRUD) estas listas de modo simplificado. Esta lógica, em conjunto com as propriedades definidas na secção anterior,

permite que, ao criar um novo controlador nos *webservices*, o WebserviceUtil o possa usar imediatamente. Neste caso, basta adicionar ao ficheiro as duas propriedades referidas de modo a que internamente o acesso a dados, a instanciação e a manipulação das listas estejam todos implementados.

A secção de temporização define métodos de conveniência para que seja possível a atualização periódica de dados.

Por último, a secção de comunicação é o ponto fulcral do WebserviceUtil. É esta secção que define os métodos de acesso ao *webservice*, assim como eventos que possam ser subscritos por outras classes de modo a que estas sejam notificadas quando existem novos dados nos *webservices* ou quando ocorrem operações CRUD sobre os mesmos. Existem ainda eventos que são acionados aquando da ocorrência de erros, tanto a aceder como a manipular os dados. Esta secção é composta por quatro subsecções: Get, Post, Put e Delete.Estas subsecções contêm uma estrutura muito semelhante, sendo compostas pelos eventos associados à subsecção, pelos métodos que permitem acionar os eventos e pelo método de acesso/manipulação dos dados. A figura que se segue ilustra a subsecção Get:

```
public static event EventHandler<WsEventArgs> GettingData;
private static void OnGettingData(Type type) { /*...*/ }

public static event EventHandler<ReceivedEventArgs> GotData;
private static void OnGotData<T>(List<T> list, string wsOutput) { /*...*/ }

public static event EventHandler<ErrorEventArgs> GotError;
private static void OnGotError(Type type, string message, string wsOutput) { /*...*/ }

public static void GetFromServer<T>()
{
    var url = GetUrl<T>(UrlType.Get);
    HttpUtil.Get<List<T>>(BaseAddress, url,
        successAction: (list, wsOutput) =>
        {
            list = list.Where(t =>
                (t as EntityBase).IsRemoved == false).ToList();
            OnGotData(list, wsOutput);
            AddToList<T>(list);
        },
        errorAction: (message, wsOutput) =>
        {
            OnGotError(typeof(T), message, wsOutput);
        });
    OnGettingData(typeof(T));
}
```

Figura 38 - Subsecção Get do Webserviceutil.

Esta subsecção define três eventos: o primeiro é despoletado antes do pedido aos *webservices* – GettingData; o segundo é invocado quando os dados são efetivamente recebidos – GotData; o último apenas ocorre quando houver uma falha na interação com os *webservices*. No caso do Get as falhas possíveis estão relacionadas com a comunicação. No entanto, para outras

subsecções podem ocorrer outros tipos de falhas: conflito ao persistir, tentativa de persistir um objeto inválido, objeto não encontrado, etc.

Na figura é apresentado após a secção de eventos o método que obtém os dados do servidor – GetFromServer. O método recebe um parâmetro genérico que indica o tipo de entidades a obter. Com base nesse parâmetro é derivado o Uniform Resource Locator (URL) que deve ser acedido para obter os dados pretendidos. Este URL é calculado utilizando *Reflection*, tendo em conta o tipo de pedido – Get – assim como o tipo de entidade que se pretende obter. Internamente é procurada uma propriedade com o nome EntityController[nome da entidade] e é derivado o URL.

A chamada aos *webservices* propriamente dita é realizada por uma classe denominada HttpUtil. Esta classe faz parte dos utilitários desenvolvidos – com o *namespace* Tools.Utils no projeto Tools.Utils.RestSharp. Este utilitário contém métodos que permitem consumir serviços REST de forma automatizada e assíncrona. Para garantir a funcionalidade de assincronismo, os métodos da classe HttpUtil recebem como parâmetros, em adição ao URL/URI (Unique Resource Identifier), acções (vulgo *callbacks*) a executar em caso de sucesso, erro ou ambos. Esta última ação é opcional, e serve para poder estipular uma ação que é executada independentemente da ocorrência de erro ou não. O assincronismo, assim como a parte da ligação HTTP, é implementado com recurso à biblioteca RestSharp.

Na figura seguinte encontra-se ilustrada a mecânica para um pedido Get:

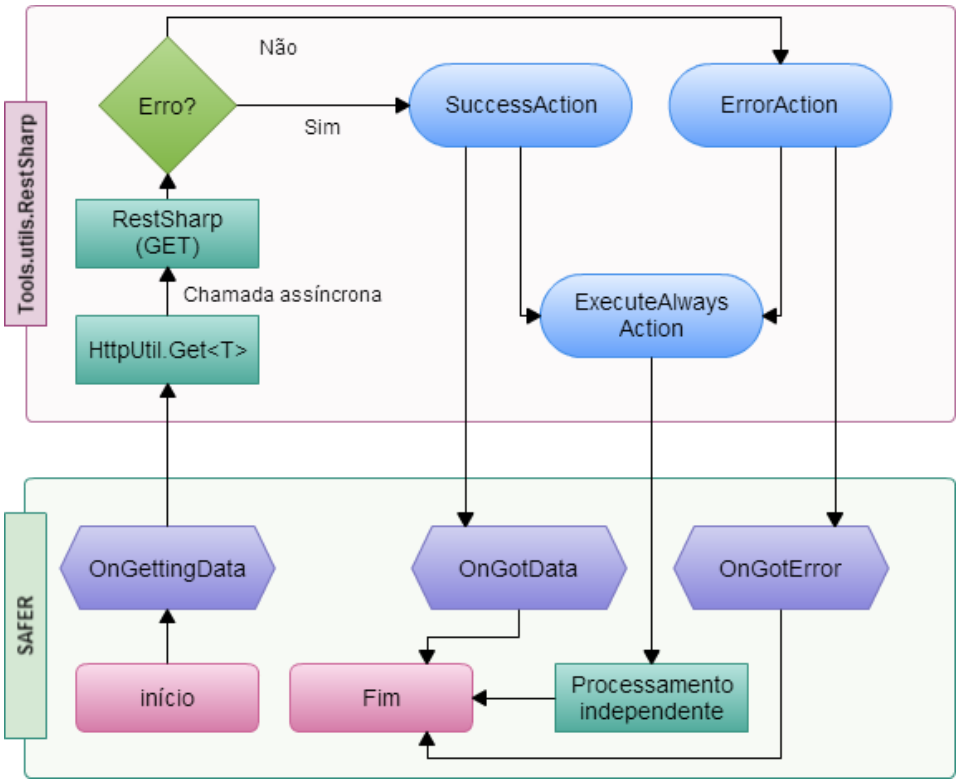


Figura 39 - Mecânica de um pedido Get do Webserviceutil.

O esquema apresentado para um pedido Get é idêntico para os restantes pedidos Post, Put e Delete. Com o auxílio do WebserviceUtil a gestão das ligações aos *webservices* pode ser realizada de modo simples, uma vez que é um ponto central de acesso. Os métodos que despoletam os eventos (OnGettingData, OnGotData, OnGotError e semelhantes para os restantes métodos) são bons sítios para realizar operações de *logging*. À semelhança do projeto que disponibiliza os *webservices*, também no cliente a biblioteca utilizada para *logging* é a log4net.

O próximo capítulo descreve as conclusões deste trabalho, revelando ainda o trabalho futuro neste projeto.

5 Conclusão e trabalho futuro

Inserindo-se num contexto empresarial, e tratando-se de um projeto em desenvolvimento, este projeto não terminou com a entrega deste documento. Conseguiu-se com o SAFER uma arquitetura para o sistema que é escalável, e dá resposta à logica das Forças de Segurança.

O sistema implementado envolve uma arquitetura cliente-servidor, utilizando tecnologias Microsoft para o desenvolvimento, tendo em conta padrões de desenvolvimento e boas práticas de desenho arquitetural. A aplicação servidora, construída sobre a plataforma ASP .NET MVC WebAPI, disponibiliza um conjunto de *webservices* capaz de gerir centralmente os dados do sistema. Esta aplicação encontra-se implementada de um modo genérico, de modo a facilitar futuro desenvolvimento neste sistema. A aplicação cliente consiste num *plugin WinForms* que faz uso dos *webservices* criados, e disponibiliza um conjunto de formulários. Para a criação de formulários semelhantes no modo de funcionamento, implementou-se um sistema de formulários reutilizáveis, através de uma base comum partilhada entre os mesmos. Transversalmente a estes dois projetos, foram desenvolvidas bibliotecas de utilitários e suporte ao projeto. Estas bibliotecas podem ser utilizadas em praticamente qualquer projeto.

Sendo um projeto ainda em desenvolvimento, e sendo a intenção implementar em clientes da Tecmic, são necessários testes exaustivos de aceitação de forma a estabilizar o produto, após a conclusão da implementação do mesmo. Este paradigma cliente-servidor não é utilizado em todos os *plugins,* estando no entanto planeada a migração de outros *plugins* do 4 Forces para fazer uso do paradigma ciente-servidor.

Em relação ao SAFER, são várias as direções que a Tecmic pretende explorar. Por exemplo, em relação à gestão de missões, existem já janelas e controladores implementados, mas esta parte ainda se encontra num estado de implementação mais "verde" do que o resto do *plugin*. Pretende-se integrar na janela principal um mapa, de modo a permitir a visualização rápida da localização das ocorrências abertas, da posição atual dos agentes e veículos e dos pontos de interesse geográfico. Apesar de haver já algum trabalho realizado nesse sentido, ainda se

encontra em progresso na altura da escrita deste documento. Em relação ao servidor, está prevista a implementação de um sistema de limpeza de dados removidos. Os dados neste momento não se encontram a ser removidos, de modo a preservar histórico de relações entre objetos. Este sistema de limpeza irá limpar esses dados, movendo os dados removidos para um sistema à parte. Outro mecanismo que poderá ser implementado no servidor é o de paginação. Neste momento todos os dados (não removidos) estão a ser transmitidos em cada pedido. Está prevista a implementação de paginação de resultados no servidor ou semelhante (obter dados mais recentes do que a data do último pedido, etc.).

Também para o SAFER, encontra-se prevista a criação de uma aplicação móvel. Esta aplicação surge da necessidade constante de monitorizar a atividade dos agentes no terreno. Para além da monitorização, esta aplicação permitirá também que estes possam ter conhecimento imediato de possíveis alterações de contexto ou de nova informação que surja durante a execução de funções no terreno. Isto permite que nova informação possa ser delegada assim que surge e que fique disponível de imediato em qualquer local, desde que os agentes tenham consigo um dispositivo com a aplicação e tenham conetividade a uma rede de dados ou *Wi-Fi*. De igual modo, os agentes no terreno têm necessidade de comunicar/reportar o estado da situação diretamente para o sistema principal, assim como para quem está a gerir a operação em curso. A aplicação móvel destina-se a *smartphones* e, numa fase inicial, será implementada em Android, podendo mais tarde alargar-se a outras plataformas móveis, consoante um estudo de viabilidade. A sua função principal consistirá em reportar a localização do utilizador, obtida através de Global Positioning System (GPS), para um servidor para que possa ser consultada na aplicação 4 Forces, de forma a monitorizar a atividade dos agentes no terreno. Para além da localização, o telemóvel será responsável pela integração com o Vital Jacket [113], que providencia informações vitais acerca do estado do agente, tais como batimento cardíaco e ainda alarme de pânico (o tão famoso *panic button*). Estas informações são importantes pois possibilitam, de uma forma simples, saber se o agente se encontra exposto a situações de perigo (que por natureza vão fazer disparar a frequência cardíaca). Para além da leitura da frequência cardíaca, o Vital Jacket disponibiliza o *panic button* que permite enviar um sinal de alerta de perigo pessoal de uma forma discreta visto que não é sinalizado pelos meios convencionais (diga-se, por exemplo, telefonar para um número de emergência ou utilização de sistemas de rádio no caso das FS). É ainda um processo mais rápido do que estes meios, na medida em que não necessita de um atendimento direto ou de uma confirmação e na medida em que o alarme é emitido em segundos.

Para além de reportar dados acerca do agente para o sistema principal, a aplicação móvel

poderá evoluir mais tarde para ser utilizada por agentes em funções no terreno para consultar o estado de missões a que esteja afeto, receber instruções ou notificações acerca de acontecimentos relevantes, como a alteração do estado de uma ocorrência. Esta aplicação móvel tem potencial para ser muito útil às Forças de Segurança.

Ainda sobre o SAFER, com a aplicação móvel poderá ser introduzido o conceito de comentários sobre a evolução de uma ocorrência, *upload* de fotografias do local, etc.

Como já referido nos capítulos referentes à introdução e revisão do estado da arte, o sistema no geral encontra-se muito (demasiado) orientado à ANPC. Também faz parte do trabalho futuro reestruturar o trabalho de modo a que fique mais genérico e, consequentemente, escalável para grupos de utilizadores futuros.

O estágio na Tecmic foi uma experiência enriquecedora, onde foi obtido conhecimento, experiência e foram criadas relações a nível profissional e pessoal.

Bibliografia

[1] "XTraN 4 Forces." [Online]. Available: http://www.tecmic.pt/por/4Forces/4Forces_intro.html. [Accessed: 29-Oct-2103].

[2] C. de Freitas and C. Gomez, "Análise de riscos tecnológicos na perspectiva das ciências sociais," *... , Ciências, Saúde—Manguinbos. Brazil, ...*, pp. 485–504, 1996.

[3] R. Cruz, "Protocolos de actuação em caso de emergência num estabelecimento de ensino do 1. º ciclo," 2012.

[4] "História da ANPC." [Online]. Available: http://www.proteccaocivil.pt/AutoridadeNacional/Pages/HistoriadaANPC.aspx. [Accessed: 29-Oct-2103].

[5] "ANPC - Sistema Nacional." [Online]. Available: http://www.proteccaocivil.pt/SistemaNacional/SistemaNacional/Pages/default.aspx. [Accessed: 29-Oct-2103].

[6] "ANPC - Protecção Civil." [Online]. Available: http://www.proteccaocivil.pt/SistemaNacional/ProteccaoCivil/Pages/default.aspx. [Accessed: 29-Oct-2103].

[7] G. Haddow, J. Bullock, and D. Coppola, *Introduction to emergency management*. 2007.

[8] D. Alexander, *Principles of emergency planning and management*. 2002.

[9] "NFPA 1600 standard on disaster/emergency management and business continuity programs," 2004.

[10] T. A. A. L. C. da Fonseca, "O paradigma do planeamento de emergência de protecção civil em Portugal." FEUC, 21-Dec-2010.

[11] "blueCAPE." [Online]. Available: http://joomla.bluecape.com.pt/. [Accessed: 29-Oct-2103].

[12] "Action Modulers." [Online]. Available: http://www.actionmodulers.pt/default.aspx. [Accessed: 29-Oct-2103].

[13] J. Scire, D. Strimaitis, and R. Yamartino, “A user’s guide for the CALPUFF dispersion model,” *Earth Tech, Inc*, 2000.

[14] Y. Zhou, J. Levy, J. Hammitt, and J. Evans, “Estimating population exposure to power plant emissions using CALPUFF: a case study in Beijing, China,” *Atmos. Environ.*, 2003.

[15] “Google Earth.” [Online]. Available: http://www.google.com/earth/index.html. [Accessed: 29-Oct-2103].

[16] L. P. C. Pita, C. Rossa, D. X. Viegas, F. Moreira, and C. Lopes, “Fixed and mobile fire detection and monitoring in Central Portugal—A component of COTEC Project,” *For. Ecol. Manage.*, vol. 234, 2006.

[17] P. Relvas, J. Almeida, F. Rego, and F. Catry, “Estudo para implementação de um sistema de videovigilância florestal no Distrito de Viseu,” *... .) Actas do 5º Congr. Florest.*, 2005.

[18] “INOV - Solução CICLOPE.” [Online]. Available: http://www.inov.pt/pages/casestudies/case_2.php. [Accessed: 29-Oct-2103].

[19] “INOV - Inesc Inovação.” [Online]. Available: http://www.inov.pt/. [Accessed: 29-Oct-2103].

[20] I. Bosch, S. Gómez, and L. Vergara, “A ground system for early forest fire detection based on infrared signal processing,” *Int. J. Remote ...*, 2011.

[21] B. Arrue, “An intelligent system for false alarm reduction in infrared forest-fire detection,” *Intell. Syst. ...*, 2000.

[22] F. Morsdorf, E. Meier, B. Kötz, and K. Itten, “LIDAR-based geometric reconstruction of boreal type forest stands at single tree level for forest and wildland fire management,” *Remote Sens. ...*, 2004.

[23] A. Utkin, A. Lavrov, and R. Vilar, “Laser rangefinder architecture as a cost-effective platform for lidar fire surveillance,” *Opt. Laser Technol.*, 2009.

[24] J. Cosme, “História da Polícia de Segurança Pública. Das origens à actualidade,” 2006.

[25] L. Fernandes, “The early centuries of the Portuguese police system: from the Quadrilheiros to the General Intendancy of Police of the Court and of the Kingdom,” *Polic. Soc.*, 2012.

[26] A. na Antigüidade, “Capítulo I–Histórico Evolutivo,” *policiacivil.mt.gov.br*.

[27] J. C. FILHO, “Manual de direito administrativo,” 2008.

[28] P. Cavaco, “A polícia no Direito Português, hoje,” *MIRANDA, Jorge*, 2001.

[29] “Lei de Segurança Interna - Lei n.º 53/2008, de 29 de Agosto.” [Online]. Available: http://www.pgdlisboa.pt/leis/lei_mostra_articulado.php?nid=1012&tabela=leis. [Accessed: 29-Oct-2013].

[30] “Normas de Funcionamento do Gabinete Coordenador de Segurança - Decreto-Lei n.º 61/88.” [Online]. Available: http://legislacao.mai-gov.info/i/normas-de-funcionamento-do-gabinete-coordenador-de-seguranca/. [Accessed: 29-Oct-2103].

[31] “Decreto-Lei n.º 149/2001 de 7 de Maio.” [Online]. Available: http://www.dre.pt/cgi/dr1s.exe?t=dr&cap=1-1200&doc=20011416 &v02=&v01=2&v03=1900-01-01&v04=3000-12-21&v05=&v06=&v07=&v08=&v09=&v10=&v11=’Decreto-Lei'&v12=&v13=&v14=&v15=&sort=0&submit=Pesquisar. [Accessed: 29-Oct-2103].

[32] “Plataforma Nacional para a Redução de Catástrofes.” [Online]. Available: http://www.proteccaocivil.pt/SISTEMANACIONAL/COMISSAO/Pages/PlataformaNacional.aspx. [Accessed: 29-Oct-2103].

[33] “SADO: a nova plataforma para troca de informação entre os agentes da ANPC.” [Online]. Available: http://www.segurancaonline.com/noticias/detalhes.php?id=170. [Accessed: 29-Oct-2103].

[34] “CDOS Santarém - 2011-Out a Dez.pdf.” [Online]. Available: http://www.prociv.pt/CDOS/Santarem/newsletter/2011-Out a Dez.pdf. [Accessed: 29-Oct-2103].

[35] “Indra.” [Online]. Available: http://www.indracompany.com/pt-br. [Accessed: 29-Oct-2103].

[36] “Centro integrado de Segurança e Emergências de Madrid (CISEM).” [Online]. Available: http://www.indracompany.com/pt-br/sectores/seguridad-y-defensa/proyectos/6657/centro-integrado-de-segurança-e-emergências-de-madri-(cisem). [Accessed: 29-Oct-2103].

[37] “iSafety.” [Online]. Available: http://www.indracompany.com/pt-br/soluciones-y-servicios/solucion/seguranca/7115/8977/seguranca-civil-e-gestao-de-emergencias. [Accessed: 29-Oct-2103].

[38] “Ifthen Software.” [Online]. Available: https://www.ifthensoftware.com/. [Accessed: 29-Oct-2103].

[39] “IFFIRE.” [Online]. Available: https://www.ifthensoftware.com/ProdutoX.aspx?ProdID=4. [Accessed: 29-Oct-2103].

[40] “IFPROTEC.” [Online]. Available: https://www.ifthensoftware.com/ProdutoX.aspx?ProdID=7. [Accessed: 29-Oct-2103].

[41] “Gesnpolicia.” [Online]. Available: http://www.gesnpolicia.com/. [Accessed: 29-Oct-2103].

[42] “IBM News room - 2010-07-21 Memphis Police Department Reduces Crime Rates with IBM Predictive Analytics Software - United States.” 21-Jul-2010.

[43] S. Greengard, “Policing the future,” *Commun. ACM*, 2012.

[44] “Crime Mapping - Building Safer Communities.” [Online]. Available: http://www.crimemapping.com/. [Accessed: 29-Oct-2103].

[45] “Crime Reports.” [Online]. Available: https://www.crimereports.com/. [Accessed: 29-Oct-2103].

[46] “PublicEye® - Mobilizing Public Safety for Police Fire and EMS.” [Online]. Available: http://www.zco.com/publiceye/. [Accessed: 29-Oct-2103].

[47] “CrimeStar Records Management.” [Online]. Available: http://crimestar.com/default.html. [Accessed: 29-Oct-2103].

[48] “Spillman - Police Records Management Software.” [Online]. Available: http://www.spillman.com/police/records/. [Accessed: 29-Oct-2103].

[49] “ledsSuite - Public Safety & Law Enforcement Software Suite.” [Online]. Available: http://www.zuerchertech.com/products/ledssuite/. [Accessed: 29-Oct-2103].

[50] “Phoenix Police RMS.” [Online]. Available: http://www.prophoenix.com/products/police-rms.aspx. [Accessed: 29-Oct-2103].

[51] J. Robillard, L. Scott, S. Bolish, and R. Sambrook, “Commercial emergency management software: evaluation methods and findings,” *Off. GeoIntegration, Air Force Sp. ...*, 2007.

[52] W. Royce, “Managing the development of large software systems,” *Proc. IEEE WESCON*, 1970.

[53] M. Cusumano and S. Smith, “Beyond the waterfall: Software development at Microsoft,” 1995.

[54] C. Weisert, “There’s no such thing as the Waterfall Approach!(and there never was)',” *... -muodossa< URL http//www. idinews. com/waterfall. ...*, 2003.

[55] S. McConnell, *Code complete*. 2004.

[56] D. Parnas and P. Clements, “A rational design process: How and why to fake it,” *Softw. Eng. IEEE ...*, 1986.

[57] “Microsoft .NET Framework.” [Online]. Available: http://www.microsoft.com/net. [Accessed: 29-Oct-2103].

[58] “Microsoft Visual Studio.” [Online]. Available: http://www.microsoft.com/visualstudio/eng/visual-studio-2013. [Accessed: 29-Oct-2103].

[59] “Microsoft.” [Online]. Available: http://www.microsoft.com/en-us/default.aspx. [Accessed: 29-Oct-2103].

[60] “Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST).” [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Accessed: 29-Oct-2103].

[61] “Simple Object Access Protocol (SOAP) 1.1.” [Online]. Available: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. [Accessed: 29-Oct-2103].

[62] H. Wesenberg, “REST versus SOAP as Architectural Style for Web Services,” *pdf.aminer.org*.

[63] T. O’Reilly, *REST vs. SOAP at Amazon*. 2003.

[64] M. Bigolin, “REST x SOAP Análise e implementação de web services,” *saloon.inf.ufrgs.br*.

[65] P. J. Leach, T. Berners-Lee, J. C. Mogul, L. Masinter, R. T. Fielding, and J. Gettys, “Hypertext Transfer Protocol -- HTTP/1.1.”

[66] “Microsoft ASP.NET MVC 4.” [Online]. Available: http://www.asp.net/mvc/mvc4. [Accessed: 29-Oct-2103].

[67] “Microsoft ASP.NET Web API.” [Online]. Available: http://www.asp.net/web-api. [Accessed: 29-Oct-2103].

[68] “Empty ASP.NET Web API Project Template.” [Online]. Available: http://visualstudiogallery.msdn.microsoft.com/a989a149-4bc3-4292-ac8a-5101ee1722d7. [Accessed: 29-Oct-2103].

[69] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, “Comparison of JSON and XML Data Interchange Formats: A Case Study.,” *CAINE*, 2009.

[70] D. Crockford, “JSON: The fat-free alternative to XML,” *Proc. XML*, 2006.

[71] “NHibernate Forge.” [Online]. Available: http://nhforge.org/. [Accessed: 29-Oct-2103].

[72] “Hibernate - JBoss Community.” [Online]. Available: http://www.hibernate.org/. [Accessed: 29-Oct-2103].

[73] “NuGet Gallery.” [Online]. Available: http://www.nuget.org/. [Accessed: 29-Oct-2103].

[74] “Microsoft SQL Server.” [Online]. Available: http://www.microsoft.com/en-us/sqlserver/default.aspx. [Accessed: 29-Oct-2103].

[75] “Oracle.” [Online]. Available: http://www.oracle.com/index.html. [Accessed: 29-Oct-2103].

[76] “Microsoft Access – database software and applications.” [Online]. Available: http://office.microsoft.com/en-au/access/. [Accessed: 29-Oct-2103].

[77] “Firebird.” [Online]. Available: http://www.firebirdsql.org/. [Accessed: 29-Oct-2103].

[78] “PostgreSQL.” [Online]. Available: http://www.postgresql.org/. [Accessed: 29-Oct-2103].

[79] “IBM - DB2 database software.” IBM Corporation, 08-Oct-2013.

[80] “MySQL.” [Online]. Available: http://www.mysql.com/. [Accessed: 29-Oct-2103].

[81] “SQLite.” [Online]. Available: http://www.sqlite.org/. [Accessed: 29-Oct-2103].

[82] “Apache log4net:” [Online]. Available: http://logging.apache.org/log4net/. [Accessed: 29-Oct-2103].

[83] “NHibernate.Mapping.Attributes.” [Online]. Available: http://sourceforge.net/projects/nhcontrib/files/NHibernate.Mapping.Attributes/. [Accessed: 29-Oct-2103].

[84] “Fluent NHibernate.” [Online]. Available: http://www.fluentnhibernate.org/. [Accessed: 29-Oct-2103].

[85] M. Fowler, “Fluent interface,” *Internetis http//martinfowler. com/bliki/FluentInterface. …*, 2005.

[86] J. Miller, “Design For Convention Over Configuration,” *Microsoft) Retrieved April*, 2009.

[87] N. Chen, “Convention over configuration,” *h ttp//softwareengineering. vazexqi. com/files/pattern. …*, 2006.

[88] J. Sobel and D. Friedman, “An introduction to reflection-oriented programming,” *Proc. Reflect.*, 1996.

[89] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994, p. 416.

[90] M. Fowler, “Inversion of control containers and the dependency injection pattern,” 2004.

[91] A. Ferrate, A. Surya, D. Lee, M. Ohye, and P. Carff, “Building Web Apps for Google TV,” 2011.

[92] R. Vacek, S. Watkins, C. Morris, and D. Keller, “Improving the Drupal User Experience,” *Code4Lib J.*

[93] “StructureMap.” [Online]. Available: http://docs.structuremap.net/. [Accessed: 29-Oct-2103].

[94] “Patterns & practices - Unity.” [Online]. Available: http://unity.codeplex.com/. [Accessed: 29-Oct-2103].

[95] “Castle Windsor - Castle Project.” [Online]. Available: http://docs.castleproject.org/(X(1)S(vep5ob455tbfch45yybei055))/Default.aspx?Page=MainPage&NS=Windsor&AspxAutoDetectCookieSupport=1. [Accessed: 29-Oct-2103].

[96] “Ninject - Open source dependency injector for .NET.” [Online]. Available: http://www.ninject.org/. [Accessed: 29-Oct-2103].

[97] “Autofac.” [Online]. Available: http://autofac.org/. [Accessed: 29-Oct-2103].

[98] J. Kurtz, “Controllers, Dependencies, and Managing the Database Unit of Work,” *ASP. NET MVC 4 Web API*, 2013.

[99] J. Dentler, “NHibernate 3.0 Cookbook,” 2010.

[100] “The Repository Pattern (MSDN).” [Online]. Available: http://msdn.microsoft.com/en-us/library/ff649690.aspx. [Accessed: 29-Oct-2103].

[101] “Martin Fowler - P of EAA: Repository.” [Online]. Available: http://martinfowler.com/eaaCatalog/repository.html. [Accessed: 29-Oct-2103].

[102] “DGTerritório - CAOP em vigor.” [Online]. Available: http://www.dgterritorio.pt/cartografia_e_geodesia/cartografia/carta_administrativa_oficial_de_portugal__caop_/caop_em_vigor/. [Accessed: 29-Oct-2103].

[103] “NHibernate.Envers.” [Online]. Available: http://www.nuget.org/packages/NHibernate.Envers/. [Accessed: 29-Oct-2103].

[104] P. Avgeriou and U. Zdun, “Architectural Patterns Revisited–A Pattern,” 2005.

[105] E. Rescorla, “HTTP Over TLS.”

[106] “MS TechNet - Basic Authentication.” [Online]. Available: http://technet.microsoft.com/en-us/library/cc784037%28v=WS.10%29.aspx. [Accessed: 29-Oct-2103].

[107] R. T. Fielding, T. Berners-Lee, and H. Frystyk, “Hypertext Transfer Protocol -- HTTP/1.0.”

[108] “MSDN - Managing Users by Using Membership.” [Online]. Available: http://msdn.microsoft.com/en-us/library/tw292whz%28v=vs.100%29.aspx. [Accessed: 29-Oct-2103].

[109] “Component Factory - Krypton Toolkit.” [Online]. Available: http://www.componentfactory.com/product?id=3. [Accessed: 29-Oct-2103].

[110] “Component Factory.” [Online]. Available: http://www.componentfactory.com/. [Accessed: 29-Oct-2103].

[111] “RestSharp - Simple REST and HTTP Client for .NET.” [Online]. Available: http://restsharp.org/. [Accessed: 29-Oct-2103].

[112] “DockPanel Suite - The .NET WinForms Docking Library.” [Online]. Available: http://dockpanelsuite.com/. [Accessed: 29-Oct-2103].

[113] “Vital Jacket.” [Online]. Available: http://www.vitaljacket.com/. [Accessed: 29-Oct-2103].

Anexo I – Análise de requisitos

De seguida é apresentada a análise de requisitos para o sistema XTraN 4 Forces. Este documento encontra-se dividido em duas secções, cada uma dedicada a um dos módulos implementados.

XTraN 4 Forces Server Application (webservices)

Requisitos funcionais	**Prioridade**
O sistema gera a base de dados de forma automática a partir de um conjunto de objetos Plain Old CLR Objects (POCO)	Media
O sistema é de configuração simples. A maioria das configurações são substituídas por convenções (Convention over configuration principle)	Media
O sistema realiza autenticação de clientes (API)	Media
O sistema aceita pedidos de modo RestFull (HTTP verbs)	Media
O sistema aceita pedidos Remote Procedure Call (RPC)	Alta
Os pedidos ao sistema fazem uso do conceito de Unit Of Work (rollback possível em caso de erro)	Alta
O sistema permite obter uma lista de utilizadores	Alta
O sistema deve criar automaticamente utilizadores e perfis por omissão	Media
O sistema permite CRUD sobre utilizadores	Alta
O sistema permite obter uma lista de pessoas	Alta
O sistema permite CRUD sobre pessoas	Alta
O sistema permite obter uma lista de agentes	Alta
O sistema permite CRUD sobre agentes	Alta
O sistema permite obter uma lista de equipas	Alta
O sistema permite CRUD sobre equipas	Alta
O sistema permite obter uma lista de unidades	Alta
O sistema permite CRUD sobre unidades	Alta
O sistema permite obter uma lista de grupos	Alta
O sistema permite CRUD sobre grupos	Alta
O sistema permite obter uma lista de equipamentos	Alta
O sistema permite CRUD sobre equipamentos	Alta
O sistema permite obter uma lista de ocorrências	Alta
O sistema permite CRUD sobre ocorrências	Alta

O sistema permite obter uma lista de missões	Alta
O sistema permite CRUD sobre missões	Alta
O sistema permite obter uma lista pontos de interesse geográfico	Alta
O sistema permite CRUD sobre pontos de interesse geográfico	Alta

Requisitos tecnológicos	**Prioridade**
O sistema permite operar por HTTP e por HTTPS	Alta
O sistema é desenvolvido em C# .NET	Alta
O sistema é desenvolvido no IDE Visual Studio 2012	Baixa
O sistema permite utilizar SQlLite como sistema de gestão de bases de dados (SGBD)	Alta
O sistema permite utilizar SQL Server 2008 como sistema de gestão de bases de dados (SGBD)	Média
O sistema pode ser alojado em servidor IIS	Média
O sistema usa NHibernate para Persistance Ignorance e ORM	Média
O sistema deve usar Dependency Injection para libertar referências	Média
O sistema devolve dados no formato JSON	Média

XTraN 4 Forces Desktop Application (plugin)

Requisitos funcionais	**Prioridade**
O sistema permite autenticação de utilizadores	Alta
O sistema permite a visualização da informação de utilizadores	Alta
O sistema permite CRUD sobre utilizadores	Alta
O sistema permite pesquisar utilizadores	Média
O sistema permite visualizar a informação de pessoas	Alta
O sistema permite CRUD sobre a informação de pessoas	Alta
O sistema permite pesquisar pessoas	Média
O sistema permite a visualização de informação de agentes	Alta
O sistema permite CRUD sobre agentes	Alta
O sistema permite pesquisar agentes	Média
O sistema permite o agrupamento de agentes em equipas	Alta
O sistema permite a visualização de informação de equipas	Alta
O sistema permite CRUD sobre equipas	Alta
O sistema permite pesquisar equipas	Média
O sistema permite agrupar equipas em unidades	Alta
O sistema permite visualizar informação de unidades	Alta
O sistema permite CRUD sobre unidades	Alta
O sistema permite pesquisar unidades	Média
O sistema permite agrupar unidades em grupos	Alta
O sistema permite visualizar informação de grupos	Alta
O sistema permite CRUD sobre grupos	Alta
O sistema permite pesquisar grupos	Média

O sistema permite a visualização de informação de equipamentos	Alta

O sistema permite CRUD sobre informação de equipamentos	Alta
O sistema permite pesquisar equipamentos	Média
O sistema permite a visualização do estado de ocorrências	Alta
O sistema permite CRUD de ocorrências	Alta
O sistema permite a pesquisa de ocorrências	Média
O sistema permite atribuir um agente reponsável a ocorrências	Alta
O sistema permite atribuir uma missão a ocorrências	Alta
O sistema permite a visualização do estado de missões existentes	Alta
O sistema permite CRUD de missões	Alta
O sistema permite pesquisar missões	Média
O sistema permite a atribuição de recursos a missões	Alta
O sistema permite a atribuição de tarefas a missões	Alta
O sistema permite atribuir um agente reponsável a missões	Média
O sistema permite vizualisar informação geográfica referente a pontos estratégicos	Alta
O sistema permite visualizar num só mapa a posição de agentes, ocorrências, missões e ou pontos estratégicos	Média

Requisitos tecnológicos	**Prioridade**
O sistema é desenvolvivo em C# .NET	Alta
O sistema é desenvolvivo no IDE Visual Studio 2012	Baixa
O sistema é adicionado à aplicação XTraN 4 Forces no formato de plugin	Alta
O sistema utiliza a aplicação servidora para obtenção de dados	Alta
Para representação de informações geográficas utiliza-se o SharpMap	Alta

Anexo II – Atividades na Tecmic

No enquadramento do projeto, foi realizado um conjunto de ações que marcaram o percurso percorrido na Tecmic até à data de entrega deste documento. Tais ações encontram-se descritas na tabela seguinte (este plano inclui tarefas realizadas ao longo dos cinco primeiros meses, os quais não foram dedicados ao 4 Forces).

Ação	Resultado esperado
Integração com a empresa, equipa de desenvolvimento e tarefas iniciais.	
Instalação de *software* e criação de uma base de trabalho.	Ambiente de desenvolvimento equipado com *software*, extensões e ferramentas que facilitem o desenvolvimento do projeto.
Integração intraempresarial.	Conhecimento dos elementos da equipa de desenvolvimento e restantes áreas de negócio da empresa e ambientação com os produtos Tecmic (XTraN Passenger, XTraN Web, iZiTraN, Siga, Simor, XTraN 4 Forces, etc.).
Projeto da Empresa da Eletricidade da Madeira	
Ambientação com o projeto da Empresa da Eletricidade da Madeira (EEM).	Obtenção de conhecimento acerca da estrutura da solução da EEM, modelo de domínio, e projetos relevantes.
Ambientação com a aplicação terminal da EEM.	Familiarização com a aplicação terminal existente (em desenvolvimento), de forma a conseguir trabalhar sobre a mesma nos meses seguintes.
Criação de um formulário dinâmico.	Obtenção de conhecimentos acerca do funcionamento da biblioteca de serialização e criação de formulários dinâmicos. Criação de um formulário dinâmico para o sistema de gestão de equipas e carregamento do mesmo na aplicação terminal.

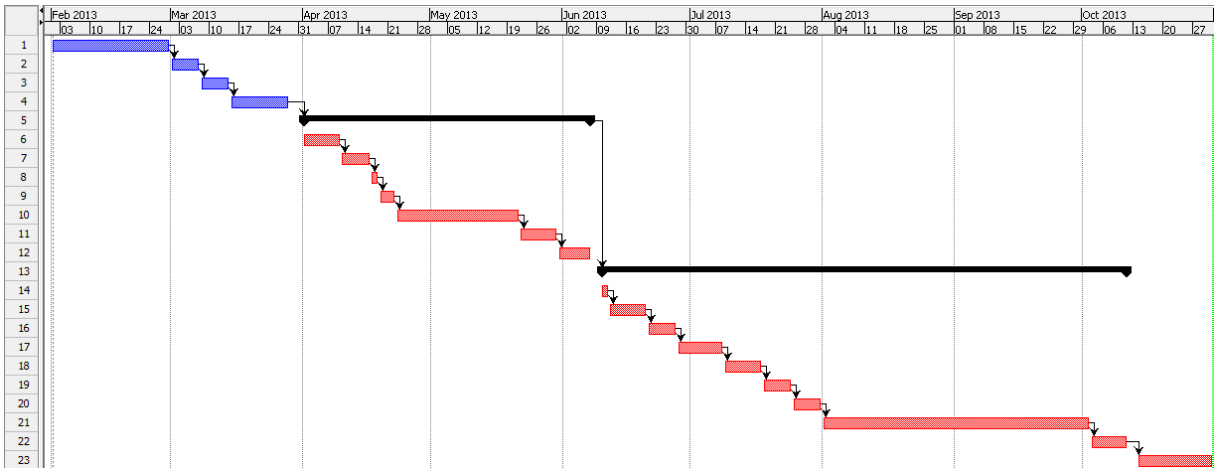
Criação de um gestor de formulários dinâmicos.	Criação de uma biblioteca reutilizável que permita o carregamento e gestão de formulários dinâmicos.
Criação de regras automáticas em formulários dinâmicos.	Criação de regras integradas com a biblioteca de formulários dinâmicos, permitindo a execução de determinadas validações aos dados ou resposta a alterações por parte do utilizador sem escrita de código adicional.
Carregamento de dados através do sistema *Generic Data.*	Compreensão do formato de serialização utilizando o sistema *Generic Data* para redução do tamanho do conteúdo de dados transportado na rede. Des-serialização dos dados provenientes de *webservices* e carregamento automático dos dados no formulário dinâmico nos campos adequados.
Profiling da aplicação terminal.	Execução de *profiling* da aplicação terminal com vista à deteção de problemas e melhoria da aplicação, tendo em conta a limitada capacidade de processamento e de memória dos terminais móveis (PDAs).
Documentação do trabalho realizado.	Criação e atualização de documentos contendo informações acerca do trabalho realizado (novas regras criadas, modo de operação etc.).
Projeto XTraN 4 Forces e SAFER Response	
Passagem de conhecimento e ambientação ao 4 Forces.	Ambientação com a estrutura da solução do XTraN 4 Forces, suas funcionalidades (de alto nível, como simulações, videovigilância, etc.) e modo de funcionamento. Esta ambientação permitiu uma melhor autonomia na implementação do projeto SAFER Response.
Criação da base de trabalho para o SAFER.	Instalação de novo *software* adicional e novas ferramentas de trabalho relevantes.
Ambientação ao Vital Jacket.	Formação acerca das características e modo de funcionamento do Vital Jacket, sistema (*hardware* e *software*) a integrar no SAFER.
Validação da API de comunicação do Vital Jacket com a plataforma Android.	Validar que a API fornecida de comunicação do Vital Jacket para Android permite a implementação das funcionalidades propostas.
Planeamento arquitetural.	Planeamento e desenho de uma arquitetura de alto nível para o sistema a desenvolver.
Análise de requisitos.	Análise de requisitos para a realização do sistema para as forças de segurança (*plugin*).

Modelo de Domínio para as Forças de Segurança (FS).	Desenho do modelo de domínio a utilizar para suportar a lógica de negócio respeitante às FS, transversalmente às várias componentes (servidor, *desktop* e *mobile*).
Criação de um *plugin*.	Criação da estrutura base de um *plugin* que possa ser carregado na aplicação.
Ambientação às bibliotecas de controlos.	Ambientação às bibliotecas Qios Dev Suite, Krypton control kit, DockPanel Suite, assim como *User Controls* existentes na solução 4 Forces, criados por outros elementos da equipa de desenvolvimento. Criação do primeiro formulário a incorporar na aplicação através do *plugin*.
Criação do servidor (*webservice*).	Criação do *webservice* de suporte às FS.
Prototipagem do *plugin*.	Desenho de protótipos não funcionais para os ecrãs a desenvolver.
Desenvolvimento do *plugin*.	Desenvolvimento do *plugin* com base nos protótipos produzidos.
Melhorias a realizar.	Após testes internos, realização de correções de eventuais *bugs* descobertos e melhorias sugeridas.
Produção de documentação.	Escrita de documentação relativa a todas as componentes implementadas (*plugin desktop*, *webservice* e aplicação Android), incluindo este documento.

Anexo III – Gráfico de Gantt

Este anexo disponibiliza o gráfico de Gantt relativo à gestão do projeto.

	Name	Duration	Start	Finish
1	Passagem de conhecimento e ambientação ao 4 Forces	20 days	2/1/13 8:00 AM	2/28/13 5:00 PM
2	Análise de requisitos	5 days	3/1/13 8:00 AM	3/7/13 5:00 PM
3	Concepção de uma arquitetura para o sistema	5 days	3/8/13 8:00 AM	3/14/13 5:00 PM
4	Criação do modelo de domínio	10 days	3/15/13 8:00 AM	3/28/13 5:00 PM
5	**Aplicação servidora (webservice)**	**50 days**	**4/1/13 8:00 AM**	**6/7/13 5:00 PM**
6	Usar DI para libertar referências e gerir validade da sessão	7 days	4/1/13 8:00 AM	4/9/13 5:00 PM
7	Implementação do padrão UnitOfWork	5 days	4/10/13 8:00 AM	4/16/13 5:00 PM
8	Implementação de um repositório genérico	2 days	4/17/13 8:00 AM	4/18/13 5:00 PM
9	Implementação de controlador base	2 days	4/19/13 8:00 AM	4/22/13 5:00 PM
10	Implementação dos restantes controladores	21 days	4/23/13 8:00 AM	5/21/13 5:00 PM
11	Implementação de segurança nos webservices	7 days	5/22/13 8:00 AM	5/30/13 5:00 PM
12	Testes aos webservices	6 days	5/31/13 8:00 AM	6/7/13 5:00 PM
13	**Criação do Plugin**	**90 days**	**6/10/13 8:00 AM**	**10/11/13 5:00 ...**
14	Desenho de protótipos para o plugin	2 days	6/10/13 8:00 AM	6/11/13 5:00 PM
15	Ambientação às bibliotecas de controlos & sistema de plugins	7 days	6/12/13 8:00 AM	6/20/13 5:00 PM
16	Implementação da estrutura PluginForm	5 days	6/21/13 8:00 AM	6/27/13 5:00 PM
17	Implementação de gestor de plugin forms	7 days	6/28/13 8:00 AM	7/8/13 5:00 PM
18	Implementação de ManagerForm	7 days	7/9/13 8:00 AM	7/17/13 5:00 PM
19	Implementação de ChooserForm	5 days	7/18/13 8:00 AM	7/24/13 5:00 PM
20	Implementação de CreatorForm	5 days	7/25/13 8:00 AM	7/31/13 5:00 PM
21	Implementação dos forms das FS	45 days	8/1/13 8:00 AM	10/2/13 5:00 PM
22	Testes aos forms	7 days	10/3/13 8:00 AM	10/11/13 5:00 PM
23	Documentação Técnica	14 days	10/14/13 8:00 AM	10/31/13 5:00 PM



Anexo IV – ReusableControls

No desenvolvimento do *plugin* SAFER para aplicação XTraN 4 Forces surgiram controlos visuais que podem ser reutilizados em projetos futuros. Como é política na Tecmic, estes controlos foram movidos para projeto próprio, originando o projeto ReusableControls. Os controlos neste projeto são SearchBar; WaitingButton; TopBorder; ExtendedDateTimePicker; QuickSplashScreen; PopupMessage; WaitingDialog; estrutura base de PluginForm desenvolvida para o SAFER (a base da estrutura pode ser reaproveitável para *plugins* futuros), incluindo PluginForm, CrudPluginForm, EntityCreator e EntityChooser). Esta estrutura encontra-se detalhada no Subcapítulo 4.5, não fazendo parte do âmbito deste anexo. Neste anexo não se pretende descrever exaustivamente os controlos desenvolvidos para o SAFER mas, sim, dar apenas alguns exemplos.

- SearchBar

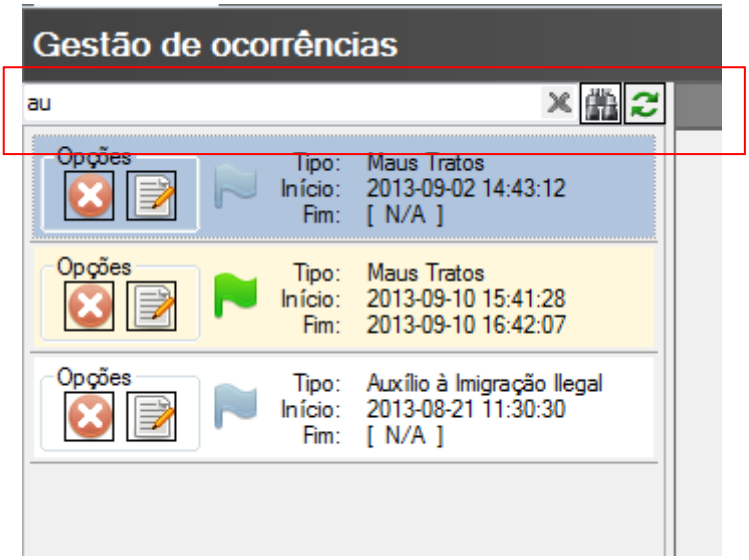


Figura 1 - Controlo SearchBar.

Disponibiliza uma barra de pesquisa pronta a utilizar e configurável para efetuar pesquisas de qualquer tipo. Possui suporte imediato ao botão ENTER, existindo no entanto um botão para ativar a pesquisa. Existe ainda um botão para limpar a pesquisa. A visibilidade de ambos botões é configurável consoante as necessidades.

- WaitingButton

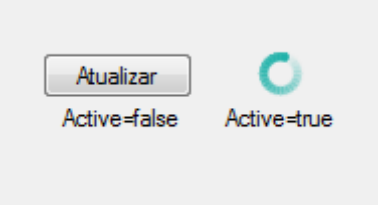


Figura 2 - Componente Waiting Button.

Desenvolvido para dar *feedback* ao utilizador quando este clica num botão que despoleta uma ação longa indicando, por um lado, que a operação se encontra em curso e, por outro, para aguardar que a mesma termine, evitando que o utilizador ao não receber *feedback* despolete múltiplas ações longas desnecessariamente. O botão é ativado normalmente com um *click*.

- TopBorder

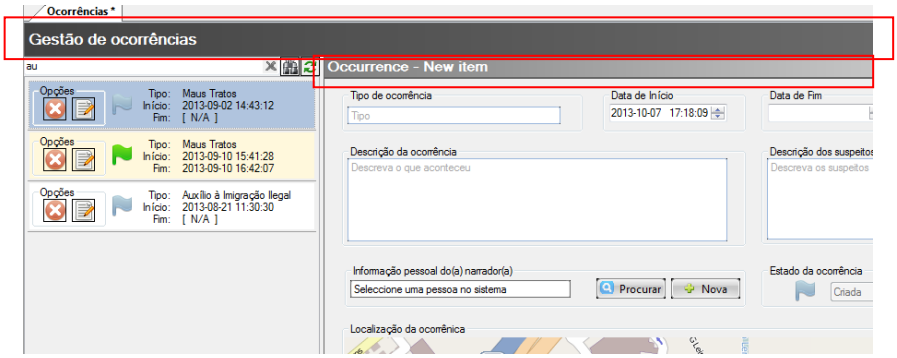


Figura 3 - Componente TopBorder.

De um modo rápido, este controlo serve apenas para definir um *header* no topo de um formulário, definindo opcionalmente um texto e as cores de fundo (suporte a gradientes).

- ExtendedDateTimePicker

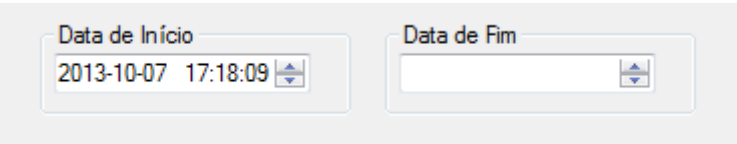


Figura 4 - Componente ExtendedDateTimePicker.

Este componente foi desenvolvido para estender o componente DateTimePicker de modo a permitir valores *null*, uma vez que, por omissão, tal não é possível.

- QuickSplashScreen

Figura 5 - Splash Screen da aplicação XTraN 4 Forces.

Este componente permite definir um *splash screen* apresentado ao utilizador enquanto o arranque inicial da aplicação é efetuado. Este componente permite controlar, de modo simples, o modo como o *splash screen* é iniciado, atualizado ou terminado.

- PopupMessage

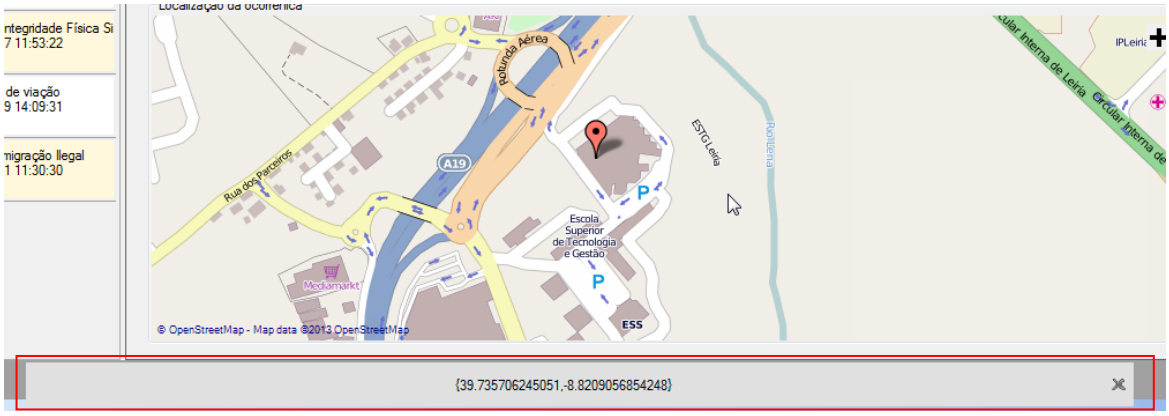


Figura 6 - PopPup message.

Este componente foi desenvolvido com a finalidade de mostrar uma mensagem de caráter secundário (não requer intervenção) do utilizador. A esta mensagem pode ser atribuída uma validade, em segundos, após a qual a mensagem fechar-se-á automaticamente, podendo o utilizador fechar a mesma manualmente, caso assim o deseje. O texto apresentado suporta seleção, de modo a ser possível efetuar ações de *copy* + *paste*.

- WaitingDialog

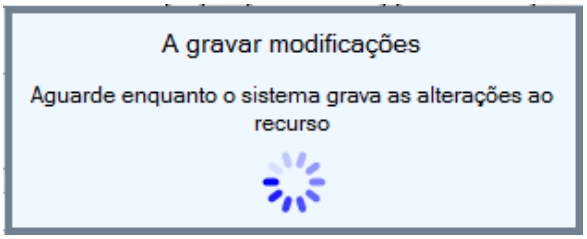


Figura 7 - Componente WaitingDialog.

Este componente é na realidade um formulário, cuja finalidade é bloquear a interface com o utilizador durante uma operação longa (*modal form*). Utilizado no SAFER, por exemplo, para dar *feedback* ao utilizador enquanto este aguarda que um objeto seja criado ou atualizado. O WaitingDialog suporta três modos de operação: espera, sucesso e erro. Estes modos podem ser facilmente comutados, sendo apenas necessário providenciar as *strings* (texto descritivo) a mostrar e as ações (Action<T>) a realizar quando o utilizador carregar nos botões – *ok* em caso de sucesso, *cancel* ou *retry* em caso de erro). Também é possível configurar se o modo de erro apresenta um botão de *retry* ou não.

Anexo V – Tools.ControlValidation

Este anexo visa dar a conhecer a utilidade da biblioteca Tools.ControlValidation, uma biblioteca implementada para o SAFER, reutilizável para outros projetos. O objetivo desta biblioteca é facilitar a validação de controlos.

A sua utilização pode ser realizada sobre quaisquer *Controls*, sendo esta biblioteca o método preferido no SAFER para validar a UI principalmente devido a duas funcionalidades principais que a biblioteca oferece: resultado da validação apresentado de forma automática (sucesso/erro) com auxílio do componente ErrorProvider; aplicação de regras automaticamente associadas a "eventos" – o que fazer em caso de erro, o que mostrar em caso de sucesso, etc.

A biblioteca está escrita de um modo *fluent*, encadeando múltiplos métodos numa sequência lógica de ações, facilitando a legibilidade e a utilização da mesma. Esta biblioteca faz uso intensivo de LINQ, *delegates* – nomeadamente *Func* e *Action* – e *lambda expressions*, tendo sido planeada cuidadosamente uma sintaxe intuitiva para a utilização da mesma.

Encontra-se definida uma interface IFormValidator que define o comportamento esperado para ser possível validar um *form*:

```
public interface IFormValidator
{
    Validator Validator { get; }
    ExtendedValidation<T> Validate<T>(T control) where T : Control;
    ExtendedValidation<T> ValidateOnce<T>(T control) where T : Control;
    ExtendedValidation<T> ValidateIf<T>(Func<T, bool> condition, T control)
        where T : Control;
    ExtendedValidation<T> ValidateIf<T>(Func<bool> condition, T control)
        where T : Control;
}
```

Figura 8 - Interface IFormValidator.

Um *form* que implemente esta interface possui condições válidas para proceder à validação de controlos. A implementação desta interface não é obrigatória, uma vez que o objeto Validator possui todos os métodos necessários para realizar a validação. No entanto, esta interface providencia um modo simples de verificar que o formulário possui capacidades de validação

(e ainda atalhos de código para as operações mais frequentes no processo de validação).

Na figura seguinte encontra-se o diagrama de classes desta biblioteca:

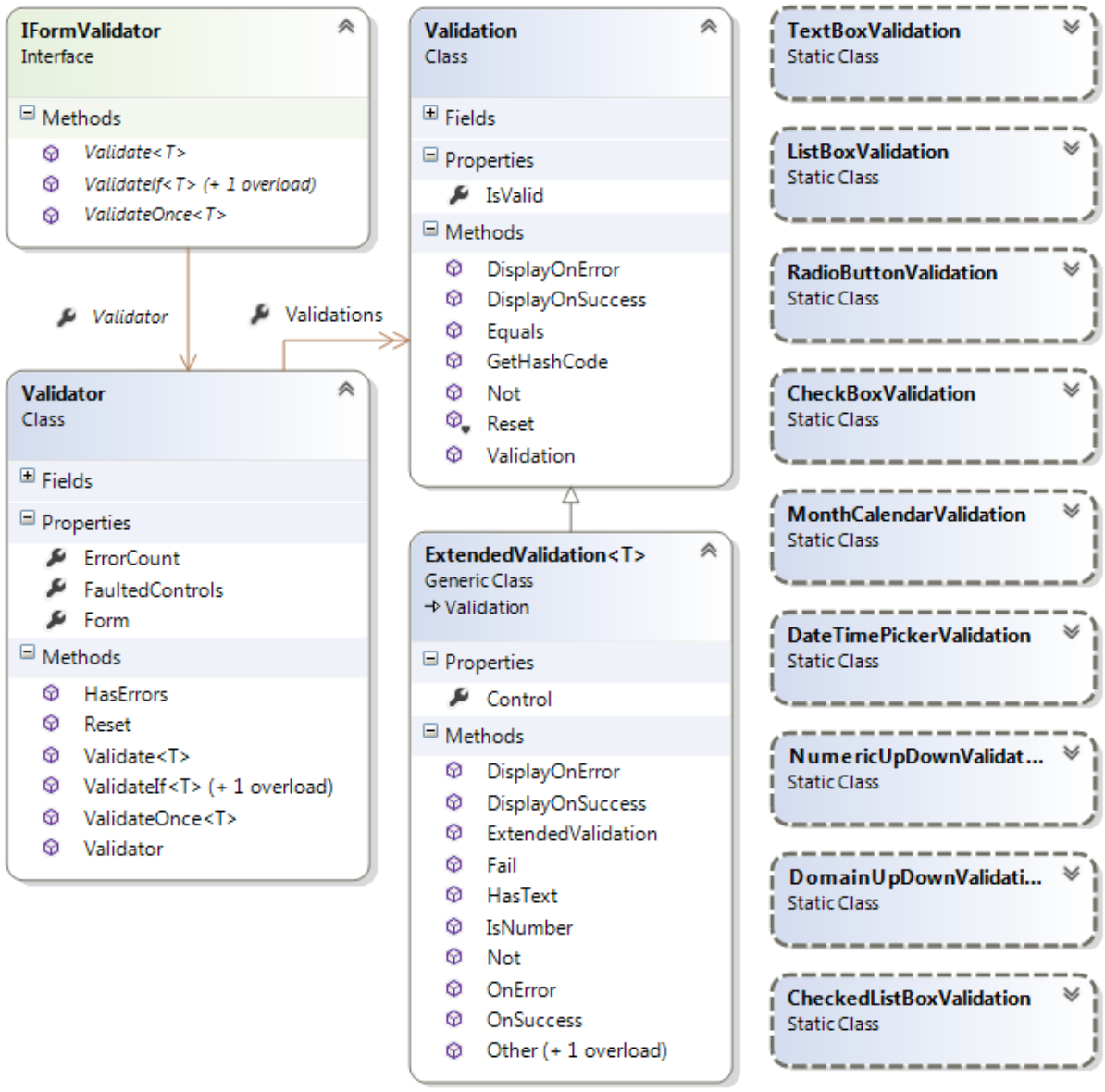


Figura 9 - Diagrama de classes Tools.ControlValidation.

O ponto de entrada na biblioteca é a classe Validator. É necessário um objeto desta classe por cada formulário que se pretenda validar, sendo disponibilizada uma interface IFormValidator para auxiliar nesta tarefa. A classe Validator retém uma lista de validações, sendo que cada validação se encontra associada a um único controlo. Ao invocar um dos métodos de validação sobre um determinado controlo, é criada uma nova validação caso esta não exista ou reaproveitada (reset) a validação existente. Este aspeto é particularmente importante, pois permite disponibilizar *properties* como ErrorCount ou FaultedControls para maior controlo sobre o processo de validação como um todo.

Todos os métodos de validação devolvem o objeto de validação – Validation ou

ExtendedValidation. O objeto Validation define o esqueleto da validação: se esta se encontra válida; qual o controlo que se encontra a ser validado; funções para mostrar uma mensagem de erro ou sucesso; operador Not. Os métodos de mostrar mensagens são implementados com recurso a controlos ErrorProvider, um para mensagens de erro e outro para mensagens de sucesso – com ícone apropriado. O operador Not é utilizado em conjunto com métodos de validação, permitindo negar o resultado da validação seguinte.

Por si só, o objeto Valiation não tem muita utilidade. Contudo, este é estendido pelo objeto ExtendedValidation, de forma a permitir estender as validações a qualquer controlo (incluindo controlos definidos pelo utilizador). Este objeto define métodos que ocorrem em caso de sucesso, erro, etc. ExtendedValidation adiciona uma camada genérica ao sistema de validações, a partir das quais é possível criar toda uma estrutura de validação. Esta biblioteca define classes de validação para vários controlos de sistema fáceis de expandir e cujo comportamento é fácil de replicar para outos controlos. Para ilustrar a utilização da biblioteca, segue-se um exemplo. Supondo o formulário OccurrencesManagerForm, para gestão de ocorrências:

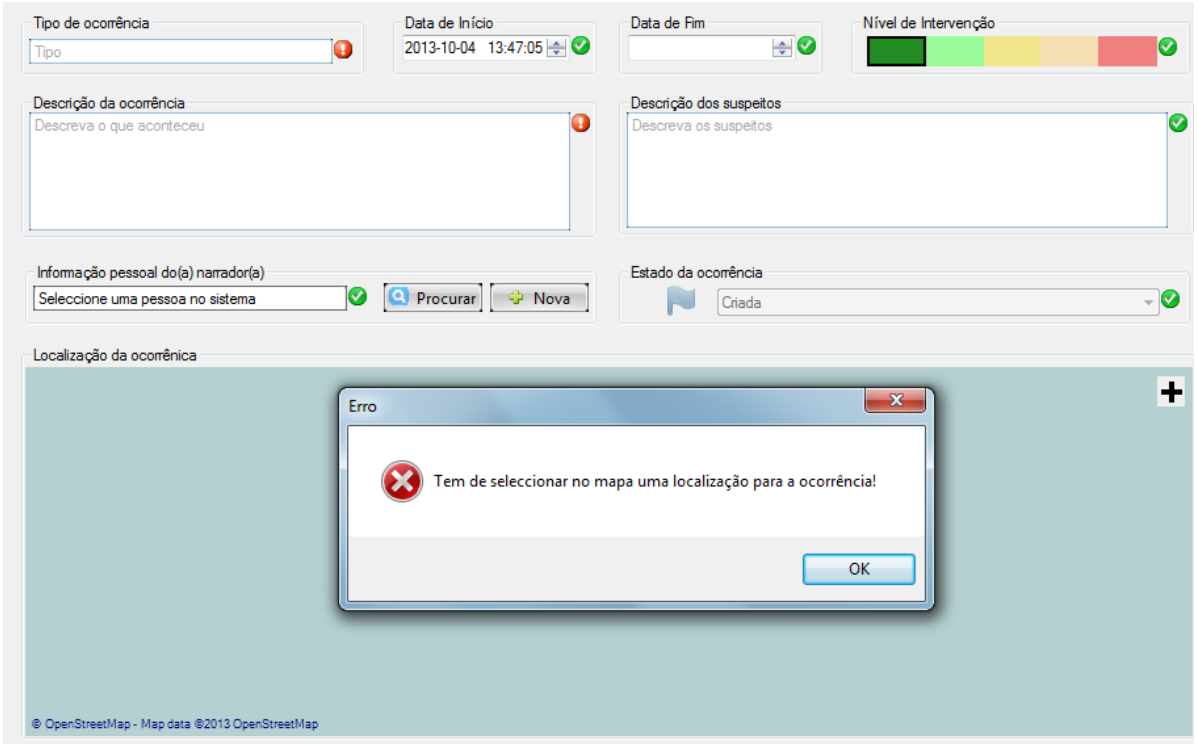


Figura 10 - Ecrã do SAFER utilizando a biblioteca Tools.ControlValidation.

Este é o resultado ao tentar inserir uma ocorrência "vazia", sem preencher os campos, carregando imediatamente no botão OK após tentar criar uma nova ocorrência no sistema.

As regras para validar este ecrã encontram-se definidas na tabela seguinte, contendo o tipo de controlo aplicado, assim como restrições e mensagens a mostrar:

Tabela 1 - Validações do ecrã de ocorrências e ações a executar.

Validação do campo	Tipo de controlo	Validação	Caso de erro
Tipo de ocorrência	QTextBox	Campo não vazio	Mensagem (ErrorProvider)
		Texto tem de ser de tipo conhecido	Mensagem (ErrorProvider)
Data de início	YacDateTimePicker	Campo obrigatório	Mensagem (ErrorProvider)
		Data não pode ser superior à data atual	Mensagem (ErrorProvider)
Data de fim	YacDateTimePicker	Não obrigatório. Data não anterior à data de início.	Mensagem (ErrorProvider – caso validado)
		Data não pode ser superior à data atual	Mensagem (ErrorProvider – caso seja validado)
Localização	SharpMapExtendedControl	Tem de selecionar localização	Lançar MessageBox a avisar

Como é possível verificar, os tipos de controlos utilizados não são controlos nativos da plataforma .NET. Alguns são disponibilizados pela biblioteca Qios Devsuite – como é o caso da QtextBox – enquanto que outros foram desenvolvidos para o SAFER. Mais informação acerca de controlos criados para o SAFER, reutilizáveis para outros projetos, pode ser consultada no Anexo IV.

Nem todas as validações presentes estão listadas na tabela. Apenas foram selecionadas algumas para demonstrar diferentes utilizações da biblioteca. Para validar estes campos, são necessárias classes e métodos que disponibilizem métodos de validações. O maior desafio na construção da biblioteca foi disponibilizar diferentes métodos para diferentes tipos de controlos, sem prejuízo da estrutura comum de validação (OnError, DisplayOnSuccess, etc.).

Este mecanismo foi implementado com recurso a *extension methods*. Apesar de inicialmente estar planeado estender a classe ExtendedValidation (herança de classes e não através *extenion methods*), adicionando os métodos específicos para cada tipo de validação, foi tomada a decisão de usar *extension methods* para evitar que subclasses de ExtendedValidation possam alterar a mecânica do processo de validação para um tipo específico. Assim, com o auxílio de *extension methods*, é possível adicionar métodos de validação apenas sem prejuízo da mecânica geral de validação.

O código utilizado para validar os campos da figura anterior é apresentado na figura seguinte:

```
// Tipo de ocorrência
Validate<TextBox>(txtType)
    .HasText() // não pode ser vazio
    .DisplayOnError(Res.BaseErrorEmptyField) // Mensagem em caso de erro
    .HasSameTextAsAny(_allowedTypes.Cast<string>(),
        ComparisonOptions.Trim | ComparisonOptions.IgnoreCase) // comparação
    .DisplayOnError(Res.OccurrenceManagerErrorInvalidType)
    .DisplayOnSuccess(Res.BaseValidField); // Caso passe em ambas validações

// Data de início. Obrigatória. Anterior à data/hora atual.
Validate<YacDateTimePicker>(datePickerStart)
    .HasNonNullValue() // tem de estar preenchida
    .DisplayOnError(Res.BaseErrorEmptyField)
    .Not().IsDateAfterCurrent() // Não pode ser depois da data atual (NOT)
    .DisplayOnError(Res.BaseErrorInvalidFutureDate)
    .DisplayOnSuccess(Res.BaseValidField);

// Data de fim. Não obrigatória, mas caso preenchida é validada.
// Não pode ser depois da data atual nem anterior à data de início
ValidateIf<YacDateTimePicker>(picker => picker.HasValue(), datePickerEnd)
    .IsDateAfter(datePickerStart.HasValue() ?
        datePickerStart.Value : DateTime.MaxValue)
    .DisplayOnError(Res.OccurrenceManagerErrorInvalidDate)
    .Not().IsDateAfterCurrent()
    .DisplayOnError(Res.BaseErrorInvalidFutureDate)
    .DisplayOnSuccess(Res.BaseValidField);

// Localização. Obrigatória.
Validate(mapLocation)
    .Other(map => map.SelectedPosition != null) // validação personalizada
    .OnError(map => SimpleMessageBox.Error(
        Res.OccurrenceManagerErrorMapNoLocation)); // Ação personalizada
```

Figura 11 - Exemplo de código de validação usando a biblioteca Tools.ControlValidation.

Sobre o código apresentado é possível realçar alguns aspetos interessantes. Em primeiro lugar, para o primeiro campo validado, apesar de este não ser do tipo TextBox, o código é válido uma vez que QTextBox estende TextBox. Mais do que uma validação pode ser encadeada. Em caso de erro, caso já haja uma mensagem a ser mostrada, por omissão esta fica, não sendo substituída. Deste modo é possível mostrar diferentes mensagens para diferentes erros. A mensagem de sucesso só será apresentada caso passe em todas as validações.

Para o segundo campo validado, salienta-se o método Not() que nega a próxima validação, invertendo o resultado da mesma. Este método foi criado para permitir lógica de negação, com um estilo de operador de negação, uma vez que as validações se encontram pensadas numa forma positiva – v.g. HasText(), IsDateBefore(), IsChecked(), Matches(). Deste modo não é necessário implementar novos métodos de validação – v.g. HasNoText(), IsDateAfter(), IsUnChecked(), FailsToMatch().

O terceiro campo validado contém uma particularidade: a não obrigatoriedade de

preenchimento. Contudo, caso esteja preenchido, deve ser validado. É este comportamento que o método ValidateIf (em vez de Validate) permite efetuar. Além do controlo a validar, este método recebe uma condição para "ativar" a validação. Caso esta condição não seja cumprida, a validação é marcada como opcional, sendo considerada sempre verdadeira (até se chamar o método Reset na validação ou chamar novamente o método Validate*, que internamente irá efetuar o *reset*).

Por último, o exemplo com menos código é talvez o mais interessante: Uma vez que para validar apenas uma condição simples não se justifica criar um *extension method* para este componente (mapa), a classe ExtendedValidation disponibiliza um método Other para executar validações personalizadas, que neste caso verifica se o mapa tem uma localização selecionada. Uma vez que o componente ErrorProvider, usado para mostrar as mensagens de sucesso ou erro ocupa espaço, optou-se por não se usar o método DisplayOnError, usando o método OnError, que permite receber uma ação a executar em caso de erro, para mostrar uma MessageBox com o erro. Note-se que o método Other pode ser utilizado para realizar todas as validações. Simplesmente não é tão apelativo sintaticamente escrever múltiplos métodos com *lambda expressions* consecutivamente do que chamar métodos "pré-fabricados" para o efeito.

Esta biblioteca desenvolvida é fortemente expansível e adapta-se às necessidades de validação dos vários controlos, incluindo controlos personalizados e é utilizada intensivamente no SAFER.