Faculdade de Engenharia da Universidade do Porto

FEUP

Sistema de notificação e reconhecimento automático de entidades em conteúdos audiovisuais

João Carlos Loureiro de Jesus Oliveira

Relatório de Projecto realizado no âmbito do Mestrado Integrado em Engenharia Informática e Computação

Orientador: Doutor Jaime S. Cardoso

Julho de 2008

Sistema de notificação e reconhecimento automático de entidades em conteúdos audiovisuais

João Carlos Loureiro de Jesus Oliveira

Relatório de Projecto realizado no âmbito do Mestrado Integrado em Engenharia Informática

Aprovado em provas públicas pelo Júri:

Presidente: Doutor João Correia Lopes

_____

Arguente: Doutor Rui Moreira

Vogal: Doutor Jaime S. Cardoso

30 de Julho de 2008

Resumo

A catalogação de material áudio-visual, no seio da indústria de conteúdos, tem dependido tradicionalmente de mão-de-obra intensiva, sendo um processo manual, lento e muito caro. Paralelamente, no universo dos conteúdos pessoais (vídeos e músicas) são notórias as dificuldades de encontrar segmentos específicos, de forma rápida e intuitiva, em grandes colecções.

Com os avanços na análise e reconhecimento de sinais áudio e vídeo, e a evolução exponencial da capacidade de processamento dos computadores, surgem condições reais para o desenvolvimento de soluções tecnológicas que ajudem no tratamento inteligente de informação, permitindo às empresas e cidadãos aumentar a eficácia e produtividade dos ambientes de catalogação e pesquisa de conteúdos.

O projecto desenvolvido nas instalações da *ClusterMedia Labs* teve como objectivo o desenvolvimento de uma solução informática de tratamento inteligente de informação audiovisual, em especial, canais de televisão ou rádio, ou ainda, colecções audiovisuais privadas. Esta solução dá ênfase à interacção e notificação dos vários utilizadores da aplicação. Orientado por esse objectivo, e partindo da capacidade já instalada da empresa, de algoritmos de reconhecimento automático, foi desenvolvida uma aplicação, no modelo cliente-servidor, responsável pela monitorização de emissões de conteúdos audio-visuais, tipicamente canais de televisão, rádio ou *Internet*, e consequente reconhecimento de determinadas entidades (pessoas, músicas, programas, publicidade, etc.) no sentido de, a partir da segmentação das emissões pelas entidades reconhecidas, gravar os conteúdos de forma organizada, e alertar os utilizadores interessados nessas mesmas entidades. A aplicação tem ainda o objectivo adicional de permitir organizar e segmentar, de forma inteligente e baseada nos conteúdos semânticos, colecções privadas ou públicas de dados audiovisuais.

Neste relatório são descritas as várias etapas e pressupostos que levaram à implementação da arquitectura da solução informática, designada por *LiveMeans*. O trabalho desenvolvido englobou a análise de requisitos e respectiva implementação dos sistemas, a análise e teste de diversas tecnologias que se apresentaram como opções para o desenvolvimento dos protótipos, conduzindo-se a um estudo de aferição das vantagens e desvantagens de cada caso. Testes de *performance* e escalabilidade fizeram ainda parte deste trabalho, de forma a avaliar a viabilidade do sistema implementado. Finalmente, foi ainda feita uma análise de possíveis implementações futuras.

Abstract

The cataloging of audiovisual material, in the heart of the industry of contents, has traditionally depended upon the use of intensive labor, being a manual, slow and expensive process. In a parallel way, in the universe of personal contents (videos and musics) there are notorious difficulties in finding specific segments in big collections, in a rapid and intuitive way.

With the advances in the analysis and recognition of audiovisual signals, and the exponential evolution of the processing capacity of computers, real conditions emerge for the intelligent handling of information, allowing companies and citizens to improve the efficiency and productivity in the environments of cataloging and content search.

The project developed in the facilities of ClusterMedia Labs had, as its main objective, the development of a computer solution responsible for the intelligent handling of audiovisual information, more specifically, television and radio channels and private audiovisual collections. This solution emphasizes the interaction and notification of the various users of the application. Guided by these objectives, and taking into consideration the existing algorithms for automatic recognition, property of ClusterMedia Labs, an application has been developed, using the client-server model, responsible for the monitoring of audiovisual emissions, typically television channels, radio networks or Internet live Streams, and consequent recognition of certain entities (speakers, musics, programs, advertisements, etc.), allowing the segmentation of the emissions by their entities, the recording of the contents in an organized way and the notification of the users interested in those entities. The application has also the additional objective of allowing the intelligent organization and segmentation of audiovisual collections, based on their semantic contents.

In this document there will be a description of the various steps and considerations that led into the implementation of the architecture of this computer solution, called *LiveMeans*. The work developed included the requirements analysis, the system implementation and the analysis and test of the various technologies that emerged as an option for developing this system. Tests of performance and scalability were also part of this project, in order to avail the viability of the implemented system. Finally, an analysis of possible future implementations was made.

Agradecimentos

Agradeço ao meu orientador, Prof. Jaime S. Cardoso, pela sua disponibilidade, pelos seus esclarecimentos, pelo seu suporte na construção deste documento e, em especial, pela ajuda disponibilizada aquando de problemas ocorridos durante o projecto.

Agradeço ao Luis Certo e ao Tiago Santos, que também realizaram o projecto nas instalações da *ClusterMedia Labs*, assim como aos funcionários Tiago Araújo e Sérgio Lopes, pelo apoio e companheirismo.

Aos meus pais, pelo constante apoio e por sempre terem acreditado em mim.

João Carlos Loureiro de Jesus Oliveira

Conteúdo

Lista de Figuras

Abreviaturas

ASP	Active Server Pages
CSV	Comma-Separated Values
DLL	Dynamic-Link Library
E-Mail	Electronic Mail
FLV	Flash Video
IDE	Integrated Development Environment
IIS	Internet Information Services
MD5	Message-Digest Algorithm 5
MMS	Multimedia Messaging Service
RTMP	Real Time Messaging Protocol
SGBD	Sistema de Gestão de Bases de Dados
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
URL	Uniform Resource Locator
XML	Extensible Markup language

Capítulo 1

Introdução

Este capítulo visa enquadrar o projecto, com uma breve descrição da empresa onde foi desenvolvido, e descrever as principais motivações e objectivos que estiveram na sua génese. Procede-se ainda a uma descrição concisa do projecto e principais contribuições do mesmo. No final, é feita uma breve apresentação dos restantes capítulos do relatório.

1.1 Enquadramento

O presente relatório insere-se no projecto final do curso Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto [FEUP08]. O projecto teve a duração de 20 semanas, de 18 de Fevereiro a 31 de Junho, nas instalações da *ClusterMediaLabs* [CML08], situada na incubadora de empresas da Universidade de Aveiro. A *ClusterMedia Labs* é uma *start-up* de base tecnológica, criada a partir da combinação do trabalho de investigação dos seus fundadores com o estímulo do prémio atribuído em 2004 pela Agência de Inovação [ADI08]. A base nuclear da Empresa acenta no desenvolvimento de algoritmos de reconhecimento automático de conteúdos audiovisuais, a partir da conjugação de duas grandes áreas científicas: Processamento digital de sinais áudio e vídeo e inteligência artificial/*machine learning*.

1.2 Motivações

As motivações que levaram à realização deste projecto prendem-se, sobretudo, com as dificuldades existentes em catalogar e segmentar grandes quantidades de conteúdos audiovisuais. Estas dificuldades são ainda mais notórias quando se fala em grandes colecções de conteúdos, tais como portais de partilha de vídeos, sendo o *YouTube* o caso mais proeminente. No entanto, mesmo no caso de pequenas ou médias

colecções privadas, são notórias as dificuldades em encontrar segmentos específicos de forma rápida e intuitiva.

Paralelamente ao problema em organizar grandes colecções de dados audiovisuais, temos ainda o caso do grande número de emissões televisivas, radiofónicas, e ainda emissões *online*, tornando-se um grande desafio efectuar a monitorização de cada emissão e a gravação de conteúdos específicos detectados nas mesmas.

De facto, nestas situações, o uso de métodos manuais deixa de ser uma opção, já que é virtualmente impossível reunir a mão-de-obra necessária para o controlo, organização, catalogação e segmentação de toda a informação audiovisual. Assim sendo, revela-se fundamental, a necessidade de desenvolver tecnologias, baseadas em inteligência artificial, que consigam efectuar estas tarefas de forma automática, com pouco recurso a tarefas manuais.

No contexto da empresa onde o projecto foi desenvolvido, sendo que a sua competência central é o desenvolvimento de algoritmos de análise de sinal de áudio e vídeo, e técnicas para *Machine Learning*, assim como o desenvolvimento de aplicações que tirem partido desses algoritmos, torna-se bastante interessante e válido, o desenvolvimento de um produto que vise resolver estas limitações da indústria de audiovisuais.

1.3 O Projecto

Inicialmente, o projecto visava o desenvolvimento de um sistema que, utilizando algoritmos de reconhecimento de áudio já existentes na empresa, permitisse uma catalogação e segmentação automática de uma colecção de conteúdos audiovisuais, tal como a existente no *YouTube* [YT08]. Posteriormente, este sistema passou a ser apenas um módulo de um projecto mais abrangente, o *LiveMeans* [LM08], já idealizado e prototipado pela *ClusterMedia Labs*, mas a necessitar de uma profunda revisão arquitectural e tecnológica. Esta redefinição do projecto, para um mais abrangente, configurou-se em duas vertentes:

- **Vertente Live** – Monitorização e Análise em tempo real, de canais de televisão, rádio, ou *Internet*, no sentido de reconhecer, com a utilização de algoritmos de reconhecimento de áudio, conteúdos audiovisuais tais como inícios e finais de programas de televisão, oradores, músicas, publicidade, linguagens, palavras chave, etc.
- **Vertente Video On Demand** – Análise de ficheiros audiovisuais, existentes numa colecção privada do utilizador, no sentido de, à semelhança da vertente *Live*, segmentar e reconhecer entidades nos conteúdos.

Em comum às duas vertentes, existe o módulo de alertas, em que os utilizadores da aplicação, interessados nos conteúdos que foram reconhecidos, são notificados, quer

pelo *front-end* gráfico, ou, caso não estejam nesse momento a utilizar a aplicação, por *e-mail* ou *sms*. Estes alertas notificam um utilizador que um determinado conteúdo, em que este está interessado, foi reconhecido num determinado canal (vertente *Live*) ou num ficheiro enviado por outro utilizador (vertente *Video On Demand*).

1.4 Objectivos

Os objectivos deste projecto baseiam-se na resolução dos problemas actuais da indústria de conteúdos audiovisuais. Assim sendo, e tendo em conta as motivações descritas na secção anterior, os objectivos definidos prendem-se com o desenvolvimento de soluções que permitam o seguinte:

- Monitorização, em tempo real, de canais de televisão e rádio, no sentido de, utilizando os algoritmos da empresa, reconhecer determinadas entidades.
- Organização e segmentação automática de uma colecção de dados audiovisuais, por conteúdos semânticos.
- Notificação dos utilizadores acerca dos conteúdos detectados, por um *front-end* gráfico ou remotamente por *sms* ou *e-mail*.

1.5 Contribuições Relevantes

No final do projecto, várias foram as contribuições resultantes para a empresa. Os objectivos foram atingidos e as contribuições resultantes são as seguintes:

- Redefinição da arquitectura e base de dados do sistema *LiveMeans*.
- Desenvolvimento, na totalidade, do *back-end* do *LiveMeans*.
- Integração do *back-end* com o *front-end* gráfico, desenvolvido por uma equipa de designers subcontratada.
- Criação de um aplicativo que, instalado em cada máquina, permite a distribuição de tarefas do *back-end* por várias máquinas, no sentido de dividir o processamento necessário para as emissões de canais televisivos.
- Integração do módulo *Video On Demand*, que permite que os utilizadores do *front-end* gráfico possam submeter novos vídeos para que o sistema os segmente por conteúdos semânticos.
- Desenvolvimento de uma *Gateway* de notificação de utilizadores, baseada numa *Gateway* de *sms* e num servidor de *SMTP*.
- Criação de um *WebCrawler* de vídeos do *YouTube* que permite, mediante a inserção de um conjunto de *keywords*, definir o número de vídeos que se pretende que sejam extraídos do servidor do *YouTube*.
- Criação de um segmentador de vídeos que permite efectuar o *Ground Truth*, ou seja, a segmentação e atribuição de *tags* de forma manual, de modo a se obter segmentos de vídeos de várias entidades e, posteriormente, criar modelos com esses segmentos.

- Adaptação e modificação profunda de bibliotecas de reconhecimento de faces, no sentido de construir uma base de dados de características faciais de cada entidade, e permitir o reconhecimento de alguém, a partir de uma nova imagem da sua face.
- Investigação e desenvolvimento de protótipos na área do reconhecimento de faces.
- Criação de uma aplicação que, baseando-se na segmentação efectuada com o segmentador de vídeos, utiliza os algoritmos da empresa para criar modelos de som. Caso se tratem de oradores, são ainda utilizadas as bibliotecas alteradas para reconhecimento de faces, no sentido de criar modelos da face das várias entidades.

1.6 Estrutura do relatório

Este relatório é composto por 7 capítulos. O primeiro capítulo é dedicado à introdução, apresentando a empresa, contexto global do problema, fazendo uma pequena descrição das soluções propostas. O segundo capítulo é uma extensão do primeiro, efectuando um estudo mais profundo sobre os problemas em causa, bem como a apresentação formal das soluções propostas e o seu faseamento.

As tecnologias envolvidas neste projecto são descritas no terceiro capítulo, em conjunto com outras tecnologias que poderiam ter sido utilizadas. São também descritas as vantagens e as desvantagens das diferentes tecnologias.

O quarto capítulo consiste na apresentação formal dos sistemas a desenvolver, a especificação da sua arquitectura e a análise dos casos de uso e requisitos. Os dois capítulos seguintes consistem na descrição da implementação dos protótipos que foram apresentados no capítulo anterior, bem como os problemas e as soluções encontradas durante essa fase.

O sétimo capítulo é dedicado à análise de resultados dos protótipos desenvolvidos, discutindo-se as *performances*, bem como as impossibilidades tecnológicas que emergiram. É ainda feita uma reflexão global sobre o projecto, abrangendo o protótipo desenvolvido, a transição para o mundo do trabalho, a integração e o impacto na empresa.

Posteriormente a estes capítulos, seguem-se as referências, bibliografia e anexos.

Capítulo 2

Revisão Bibliográfica

Neste capítulo, irá ser descrito o problema actual da indústria de conteúdos audiovisuais e, de seguida, irá ser discutido o estado de arte no que toca a algoritmos de análise de sinal audiovisual.

2.1 Problema

No quotidiano de todos nós, as funções naturais desempenhadas, como ouvir, reconhecer e interagir são vitais para a elaboração de tarefas profissionais e pessoais. Acresce que o incessante débito com que se produz informação multimédia provoca dificuldades evidentes no acesso e tratamento dessa enorme quantidade de informação dispersa, lançando grandes desafios na investigação e desenvolvimento de soluções automáticas de indexação e pesquisa de conteúdos áudio e vídeo.

O desafio de tratar de forma inteligente conteúdos audiovisuais pode ser dividido em duas classes distintas, a de conteúdos *not live* (colecções de conteúdos) e de conteúdos *live* (emissões de televisão, rádio ou *liveStreams* da Internet):

2.1.1 Conteúdos not Live

Um caso muito concreto prende-se com portais de partilha de vídeos, como o *YouTube*, *MetaCafe* [MC08], *Break* [BRE08], *DailyMotion* [DM08], *Sapo Vídeos* [SV08], etc. O caso do *YouTube* é sem dúvida o mais revelador. As estatísticas relativas a este portal são verdadeiramente impressionantes. Em Julho de 2006, o número de vídeos visualizados diariamente superava a marca dos 100 milhões, sendo que no mês de Junho desse ano, foram visualizados 2.5 biliões de vídeos [USA08]. Estes números só têm tendência a aumentar, com o rápido aumento dos vídeos disponíveis. Em Maio de 2006, foram adicionados cerca de 50 milhares de vídeos por dia enquanto que em Julho, este número subiu para 65 mil. Só no mês de Janeiro de 2008, 79 milhões de utilizadores viram mais de 3 biliões de vídeos no *YouTube* [FOR08]. Estima-se que, a 9

de Abril de 2008, a base de dados do *YouTube* continha mais de 83.4 milhões de vídeos diferentes [YTA08]. A largura de banda utilizada é de tal forma dantesca que neste momento o portal não consegue obter qualquer lucro, já que os custos relacionados com a largura de banda são estimados em cerca de 1 milhão de dólares por dia. Não é complicado acreditar nestes custos já que, a título de curiosidade, a largura de banda consumida pelo *YouTube* em 2007 foi equivalente a toda a largura de banda consumida pela Internet no ano de 2000 [TG08].

Obviamente, a tarefa de organizar toda esta quantidade de dados audiovisuais é um grande desafio. Não recorrer a algoritmos de tratamento inteligente de conteúdos deixa de ser uma opção já que é virtualmente impossível alocar tanto trabalho humano a essa tarefa. A necessidade de organizar os vídeos existentes no *YouTube* ou noutro portal de partilha de vídeos, existe por diversos motivos:

- Crescente dificuldade em pesquisar vídeos – Sendo que os resultados retornados por uma pesquisa de vídeos dependem essencialmente das *tags* inseridas pelos utilizadores, estas assumem uma importância excessiva. De facto, nem sempre as *tags* inseridas por um utilizador para classificar um determinado vídeo são as mais correctas. Podem ter sido inseridas para aumentar a popularidade de um vídeo, podem não ser suficientemente específicas, subjectivas, etc. A juntar a este problema, temos também a questão da ambiguidade inerente a uma pesquisa por palavras chave, ou seja, o facto de uma determinada *tag* poder ter significados completamente distintos em diferentes contextos. Para ilustrar este facto, podemos, por exemplo, falar do caso de um utilizador que esteja interessado em encontrar vídeos da banda *Police*, tal como é exemplificado na Figura 2.1. Ao efectuar uma pesquisa por essa palavra-chave, a maior parte dos resultados prende-se com vídeos relacionados com a polícia e não com a banda. O utilizador pode aumentar a precisão da pesquisa adicionando, por exemplo, a *tag* "*Music*" na pesquisa, mas tal solução não é a ideal já que, remove logo à partida, a possibilidade de serem encontrados vídeos que não tenham essa *tag*. Analogamente a este exemplo, existem muitos outros, tais como "*Queen*" (banda ou rainha), "*Crash Test Dummies*" (Banda ou bonecos), "*Bruce*" (*Springsteen* ou *Lee*), "ac" (*Acdc*, electricidade ou a.c. *Milan*), etc.
- Vídeos repetidos – Com um tão grande número de vídeos, é complicado controlar, de forma manual, aqueles que são repetidos. De facto, existem muitos vídeos repetidos na *Internet*, o que causa transtornos, quer aos utilizadores pela maior dificuldade e trabalho em procurar vídeos, quer ao *YouTube*, pelo uso desnecessário de recursos.
- Questões de *CopyRight* – Com o aumento da popularidade do conceito de partilha de vídeos na *Internet*, o problema de serem alojados vídeos com direitos de imagem tornou-se bastante real e problemático. É uma questão controversa já que ainda não é bem definido se a culpa fica do lado dos utilizadores que fazem *Upload* do vídeo ou do *WebSite* que os aloja. Seja como for, o *YouTube* já teve vários problemas a este nível, em especial devido à existência de conteúdos relacionados com filmes, séries, músicas, programas de televisão, etc., todos protegidos por direitos de autor. Assim sendo, torna-se impraticável a constante

monitorização manual de todos os vídeos inseridos na base de dados, no sentido de encontrar vídeos ilegais.

- Remoção de vídeos não próprios – Este problema é uma extensão do problema anterior, mas, em vez de o problema consistir na existência de vídeos protegidos por direitos de autor, o problema é a existência de vídeos não próprios para as políticas destes WebSites, tais como conteúdo erótico, pornográfico, pedófilo, ou excessivamente violento ou perverso. Mais uma vez, monitorizar todos os vídeos inseridos, no sentido de encontrar conteúdos ilegais, é algo que, devido à dimensão que este tipo de portais atingiu, deixou de ser possível fazer de forma manual.

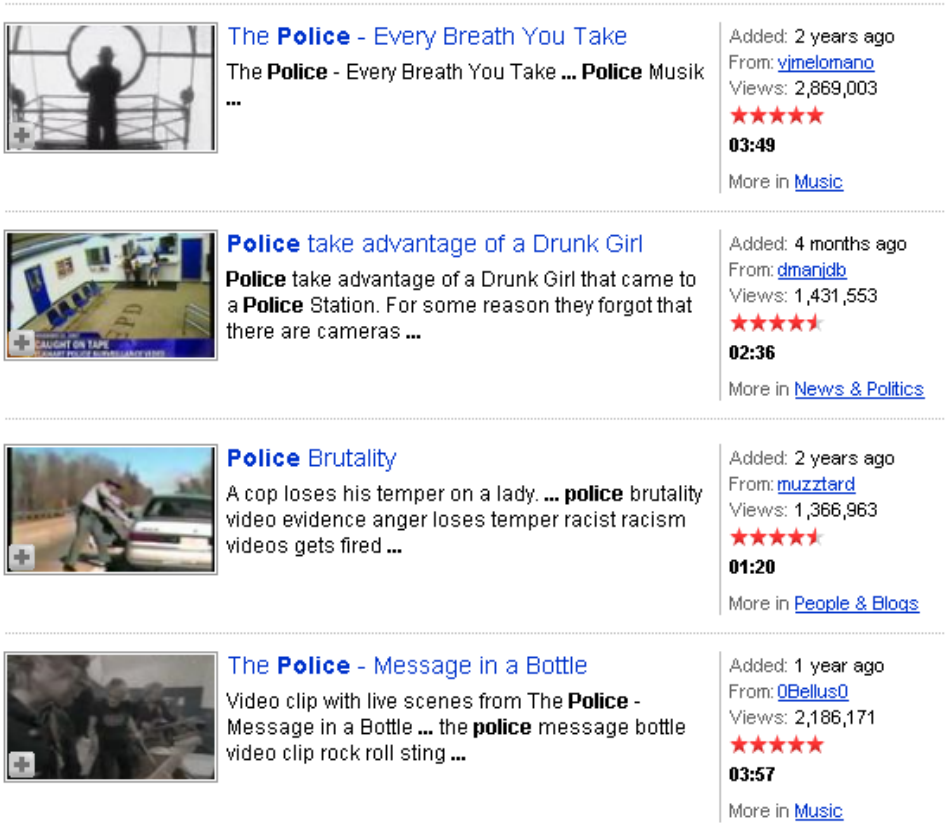

Figura 2.1 – Pesquisa no *YouTube* pela *tag* "*Police*"

Estes problemas não são exclusivos de portais de partilha de vídeo, podendo acontecer numa multiplicidade de casos, inclusive em colecções privadas de conteúdos Audiovisuais. Veja-se o caso do *iPod*, que segundo um caso de estudo do *iTunes Registry*, cerca de dois terços das músicas existentes num iPod nunca são tocadas. De resto, a lei de Pareto (80-20) também se aplica, sendo que apenas cerca de 20% de todas as músicas existentes num dispositivo são responsáveis por cerca de 80% das reproduções [DLB08].

2.1.2 Conteúdos Live

Esta vertente do problema é bastante distinta da vertente *not live*. O grande desafio prende-se com o facto de vivermos numa realidade em que existem milhares de canais de televisão, rádio ou de Internet a emitir constantemente conteúdos audiovisuais. É complicado quantificar quantos são esses canais, mas se tivermos em conta que, só em Portugal existem 23 canais (quatro terrestres e 19 por cabo) e nos Estados Unidos atinge-se quase a marca dos 100 canais [TVN08], e que, a nível de emissões de rádio, o número é ainda maior, consegue-se estimar em vários milhares, o número de canais de televisão e rádio em todo o Mundo. A juntar-se a estes, temos as emissões exclusivamente *online,* na forma de *liveStreams*.

Este cenário causa enormes dificuldades na monitorização dos conteúdos que são emitidos em todo o Mundo e, inerentemente, na execução de determinadas tarefas:

- Determinar quando é que um programa começa ou termina.
- Saber quando é que determinada personalidade aparece ou fala num canal de televisão ou rádio.
- Saber quando é que uma música ou género musical começa a tocar num qualquer canal de televisão ou rádio.
- Contabilizar tempos de publicidade em vários canais e criar estatísticas baseadas nesses dados.
- Detectar conteúdos chave ou palavras específicas na matriz de canais existentes.

Devido à já falada imensidão de canais existentes em todo o Mundo, o recurso a processos manuais para monitorizar os canais passa a ser uma tarefa praticamente impossível. Torna-se assim necessário, para ultrapassar este desafio, o desenvolvimento de tecnologias de reconhecimento automático de conteúdos em canais audiovisuais.

2.2 Como resolver o problema

Como debatido na secção anterior, a organização e monitorização de conteúdos audiovisuais em quantidade massiva, passou a ser uma tarefa apenas exequível se auxiliada por meios informáticos automatizados. Assim sendo, a solução passa pelo desenvolvimento de tecnologias capazes de reconhecer, catalogar e segmentar, de forma inteligente e autónoma, este tipo de conteúdos. Este tipo de tecnologias, se tiverem uma eficácia e precisão elevada, poderão permitir a resolução das dificuldades levantadas na secção anterior. Tais tecnologias deverão ser capazes do seguinte:

- Monitorização, em tempo real, de canais televisivos e estações radiofónicas, com o objectivo de reconhecer automaticamente os conteúdos transmitidos, tais

como programas de televisão, músicas, oradores, publicidade, palavras-chave, etc.
- Sistema de alerta a pessoas interessadas em determinados conteúdos, passíveis de serem reconhecidos nas emissões de televisão e rádio.
- Um novo paradigma de pesquisa de conteúdos audiovisuais, em especial em portais de partilha de vídeos, baseado no conteúdo semântico do vídeo e não nas *tags* inseridas pelos utilizadores
- Facilidade em ordenar, catalogar e segmentar uma colecção de dados audiovisuais, pelo seu conteúdo semântico, permitindo ainda a remoção de conteúdos repetidos, conteúdos indesejados, etc.

2.3 Estado da arte

Nesta secção irá ser feita uma descrição do estado de arte da ciência, em especial no que diz respeito às áreas de análise de sinal áudiovisual e de *Machine Learning*. Irão ser discutidos tópicos relacionados com algoritmos do reconhecimento de voz, reconhecimento de faces, recomendação de músicas e *fingerprint*.

2.3.1 Reconhecimento de oradores

O reconhecimento de oradores é a área informática que tem como objectivo reconhecer pessoas a partir da sua voz. Este tipo de sistemas extrai características de várias amostras da voz de uma pessoa, cria um modelo e usa-o para reconhecer essa voz a partir de uma nova amostra.

A história do reconhecimento de oradores teve início há cerca de quatro décadas, com um sistema muito rudimentar de filtros analógicos. Desde então, e em especial com a evolução exponencial da capacidade de processamento dos computadores, têm sido feitas bastantes melhorias nesta área [SR08].

Os sistemas actuais podem ser divididos em duas fases distintas:

- *Enrollment* – Nesta fase, o sistema analisa várias amostras da voz de um orador, criando modelos ou *templates* para o identificar.
- *Verification* – Nesta fase, o sistema analisa um novo sinal e compara-o com os modelos ou *templates* existentes, no sentido de efectuar um *match* e reconhecer um orador.

Tanto numa fase como noutra, a questão essencial prende-se com a análise do sinal de áudio das amostras da voz, a extracção de características e a utilização de técnicas de *machine learning* para criar modelos da voz de cada orador ou para fazer a comparação de uma nova amostra com os modelos já existentes.

Assim, o sistema começa por extrair as características acústicas da voz que são determinantes para diferenciar diferentes oradores. Estes padrões acústicos reflectem tanto a anatomia (tal como o tamanho e forma da garganta e boca) como comportamentos aprendidos (tom de voz, sotaque, estilo de fala, etc.). Uma questão importante, e que muitas vezes torna complicada a extracção de características, prende-se com a existência de ruído ambiente que pode degradar a *performance* dos algoritmos de análise de sinal. Para resolver estes problemas, são utilizados algoritmos de redução de ruído e de estimação de frequência.

Para a criação de modelos e *match* de novas amostras com os modelos existentes, várias são as tecnologias que se podem utilizar, tais como *Hidden Markov Models*, redes neuronais, árvores de decisão, *Support Vector Machines,* etc. [CC08]

Em termos de aplicações concretas, de reconhecimento de voz, a grande maioria tem objectivos diferentes, tais como:

- *Speech to Text* – Pretende reconhecer o que está a ser dito de forma a utilizar a informação dita pelo utilizador, ou para a guardar num documento ou para executar tarefas baseadas no que o utilizador disse. Este sistema é capaz de atingir uma eficácia relativamente elevada, ainda que não perfeita, mas exige um período de aprendizagem da voz do utilizador bastante elevado. [TAY08]
- *Voice Authentication* – Pretende substituir o típico sistema de palavra-chave por um sistema biométrico que usa a voz como factor de autenticação. De notar que neste caso, o problema passa de uma validação *um-para-muitos*, para uma validação *um-para-um*, simplificando imenso o problema. Este sistema tem, como uma das lacunas mais graves, o facto de poder ser ultrapassado com recurso a gravações de voz. [MAL04]

No que toca a aplicações de reconhecimento de várias entidades a partir de novas fontes de sinal digital, existem poucos desenvolvimentos. Um dos grandes problemas prende-se com a necessidade de criar modelos para cada nova entidade que se queira reconhecer, o que leva a uma maior complexidade e carga de processamento, já que cada nova amostra tem que ser comparada com um maior número de modelos, mesmo que existam heurísticas inteligentes para reduzir o número de comparações necessárias.

2.3.2 Reconhecimento facial

Sistemas de reconhecimento facial são aplicações informáticas que têm como objectivo identificar ou verificar a identidade de uma pessoa a partir de uma imagem digital ou de um vídeo. Estes sistemas podem variar imenso entre si mas, uma característica comum a todos, é a existência de duas fases principais:

- *Enrollment* – Nesta fase, são criados modelos faciais de uma ou várias entidades.

- *Recognition* – Posteriormente, comparam-se as características de uma nova face com os modelos faciais existentes no sentido de efectuar um *match*.

A história do reconhecimento facial remonta aos anos de 1964 e 1965, altura em que os investigadores Woody Bledsoe, Helen Chan e Charles Bisson trabalharam no sentido de utilizar computadores para reconhecer faces humanas [FR08]. Para o conseguir, o sistema, utilizando mesas digitalizadoras, detectava e extraía as coordenadas de pontos da face tais como o centro das pupilas, o centro da boca, etc. A partir dessas coordenadas, o sistema calculava vinte distâncias diferentes, tais como o comprimento da boca e dos olhos ou a distância entre as pupilas. O sistema era capaz de processar 40 fotografias por hora. Para cada fotografia eram então registadas, na base de dados, todas as distâncias calculadas. Na fase de reconhecimento, o sistema voltava a calcular as distâncias da nova fotografia, comparava com as existentes na base de dados e calculava qual aquela mais próxima. O sistema funcionava razoavelmente bem em situações controladas, ou seja, quando as fotos utilizadas consistiam sempre em fotografias frontais, iluminação semelhante, expressão facial neutra, etc. Quando isto não acontecia, em especial quando havia rotação vertical ou horizontal da face, a eficácia do sistema era francamente mais fraca. Segundo o próprio Woody Bledsoe, em 1966:

"This recognition problem is made difficult by the great variability in head rotation and tilt, lighting intensity and angle, facial expression, aging, etc. Some other attempts at facial recognition by machine have allowed for little or no variability in these quantities. Yet the method of correlation (or pattern matching) of unprocessed optical data, which is often used by some researchers, is certain to fail in cases where the variability is great. In particular, the correlation is very low between two pictures of the same person with two different head rotations." [BLE08]

Só passados cerca de 30 anos, voltaram a ser feitas grandes inovações, quando um sistema desenvolvido por Chistoph Von Malsburg e outros estudantes da Universidade de *Bochum* na Alemanha e da Universidade da Califórnia do Sul, nos Estados Unidos, revelou ter uma boa robustez, mesmo em situações em que as imagens disponíveis não eram perfeitas. O sistema conseguia ultrapassar impedimentos tais como bigodes, barbas, mudanças de penteados, óculos e até óculos de sol. [FRS08]

Na actualidade, os sistemas de reconhecimento de faces utilizam várias tecnologias para extrair as características de uma face, tais como algoritmos *Eigenface*, *fisherface, hidden markov Model* ou redes neuronais. Contudo, os grandes avanços nesta área estão a ser conseguidos graças à utilização de tecnologias *3d*. Estas tecnologias, bastante avançadas, criam modelos tridimensionais de uma face, tal como o presente na Figura 2.2, a partir de várias imagens *2d* ou de vídeos, conseguindo então, com uma nova imagem, reconhecer um certo individuo, mesmo que a imagem não esteja nas condições desejadas. [WEC06]

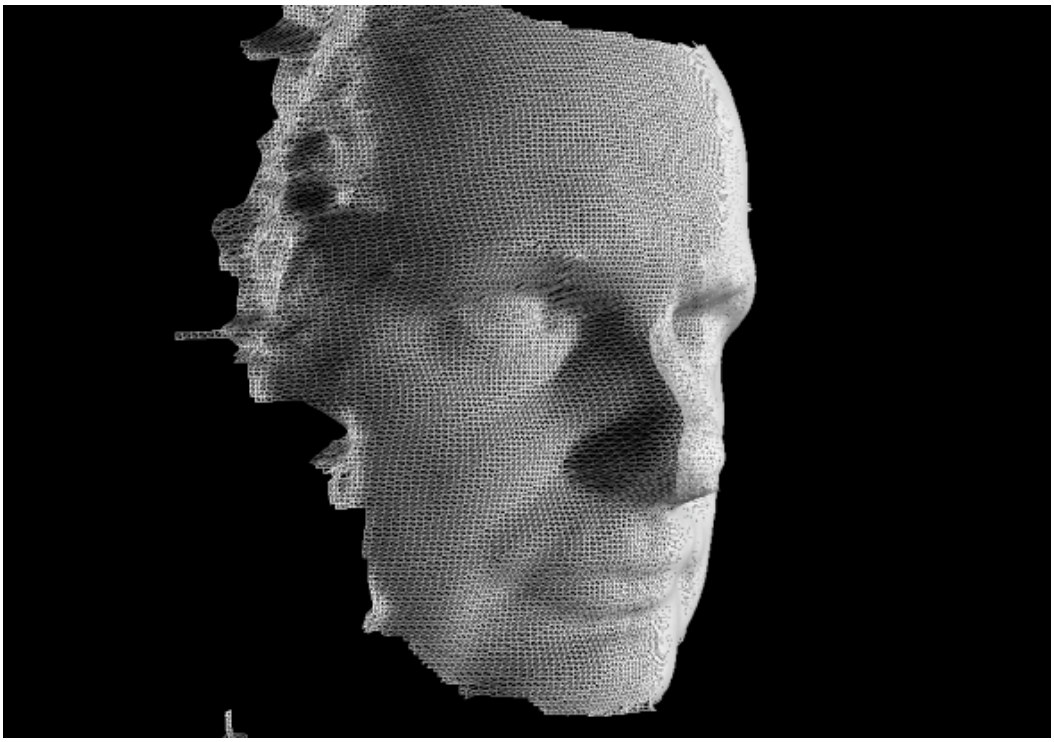

Figura 2.2 - Modelo facial utilizado para reconhecimento facial

2.3.3 Recomendação de música

A inclusão desta secção do relatório, prende-se com o facto de um dos tipos de entidades que é possível reconhecer com o sistema *LiveMeans* ser o género musical, necessitando assim, de algoritmos que sejam capazes de extrair determinadas características de uma música, no sentido de avaliar então, o género ou os géneros musicais mais prováveis. A área da recomendação de músicas está intrinsecamente ligada aos géneros musicais, já que é um dos grandes factores preponderantes na atribuição de similaridades entre as músicas.

De facto, estes sistemas de recomendação de música são cada vez mais populares, podendo-se falar de casos de sucesso como o *Pandora* [PAN08], cuja *interface* é visível na Figura 2.3, ou o *Last.Fm* [LAS08] que possuem vários milhares de utilizadores e uma base de dados musical de um volume impressionante. Contudo, estes sistemas e muitos outros possuem uma característica simples que deixa em aberto uma evolução nesta área: O facto de todas as associações entre músicas ser definida de forma colaborativa, por estatísticas baseadas na experiência de vários utilizadores, ou mesmo por utilização massiva de mão-de-obra bruta para classificar cada música e atribuir similaridades com as restantes. Como vantagens, este sistema permite uma maior qualidade dos dados. No entanto, possui grandes desvantagens, tais como a grande necessidade humana para organizar uma colecção musical por similaridades ou a dificuldade em manter e escalar o sistema.

Figura 2.3 – Interface do sistema de recomendação de música *Pandora*

Uma forma de contornar este problema, sendo, de resto, uma das áreas em que a empresa *ClusterMedia Labs* mais efectua investigação, é desenvolver algoritmos que, entre outras coisas, consigam analisar os géneros musicais de uma música e definir a similaridade existente com outras. Os algoritmos existentes para este fim utilizam diferentes abordagens para resolver o problema. Por exemplo, alguns usam a análise de géneros musicais para fazer à partida uma filtragem prévia dos possíveis resultados. Este opção pode ser benéfica, no sentido em que músicas do mesmo género, à partida tem maior probabilidade de serem similares, mas também pode ser prejudicial, já que se estão a eliminar músicas que podem ter um grande potencial de similaridade, ou porque apesar de os géneros serem diferentes, as músicas têm outras características semelhantes, ou porque o algoritmo de identificação de géneros falhou.

Para além da detecção do género musical, estes algoritmos podem utilizar imensas características, tais como a existência ou não de voz, o reconhecimento dos instrumentos musicais presentes, a análise do timbre, tempo, ritmo ou intensidade da música, etc. Posteriormente, estes parâmetros são utilizados de forma a serem comparados com os de outras músicas, calculando-se a chamada distância entre as músicas. [WII04]

Ainda de referir uma característica que é cada vez mais incorporada nestes algoritmos: o *mood*, ou seja, o sentimento que a música transmite. Os *moods* principais são o alegre e o triste, mas há vários outros possíveis tais como animado, nostálgico, saudoso, etc. Apesar de não ser o critério mais comum para pesquisa de música, pode possibilitar situações interessantes, tal como a recomendação de música de acordo com o estado de espírito do utilizador.

2.3.4 Fingerprint de dados Audiovisuais

A palavra *Fingerprint* significa impressão digital. No contexto da análise de conteúdos audiovisuais, corresponde a uma técnica que permite analisar um conteúdo

audiovisual e comparar o sinal que o caracteriza com uma determinada quantidade de conteúdos, no sentido de encontrar um segmento muito similar [KB95]. Em termos práticos permite, por exemplo, analisar o sinal de um excerto de uma música e, percorrendo o sinal das músicas presentes numa base de dados, procurar efectuar um *match* com um segmento de uma destas, identificando a que música pertence o excerto analisado. Esta técnica é utilizada para um grande número de objectivos, mas, no caso concreto do *LiveMeans*, aqueles que fazem mais sentido são os seguintes:

- No módulo *Video On Demand,* permitir, aquando do *upload* de um ficheiro por parte de um utilizador, comparar com os restantes conteúdos já existentes, no sentido de detectar a existência de um vídeo semelhante.
- No mesmo contexto do ponto anterior, permitir efectuar o *fingerprint* com uma lista de conteúdos proibidos (protegidos por *copyright* ou não próprios), no sentido de analisar se o vídeo, ou parte do vídeo que está a ser enviado para o servidor, possui conteúdos ilegais.
- Já no módulo *Live,* reconhecer uma determinada música por *fingerprint* de parte da mesma, ou reconhecer que um determinado programa começou através de *fingerprint* da música do genérico do programa.
- Também no módulo *Live*, reconhecer, por *fingerprint* de parte do sinal de vídeo, o símbolo do canal no canto da imagem, de forma a saber se a emissão está a emitir programas ou publicidade.

Capítulo 3

Revisão Tecnológica

Neste capítulo são apresentadas as diferentes tecnologias que foram ponderadas como opções válidas para a implementação de cada protótipo.

3.1 Linguagens de programação

Foram várias as linguagens de programação que foram utilizadas para contribuir, directa ou indirectamente para o projecto. Segue-se uma descrição de todas elas.

3.1.1 Ruby

Foi utilizado inicialmente, devido ao facto do antigo protótipo do *LiveMeans*, existente na empresa, ter bastantes componentes desenvolvidas nesta linguagem (utilizando-se a *frameWork Ruby On Rails*). Assim, as primeiras duas semanas de desenvolvimento do projecto foram dedicadas à aprendizagem desta linguagem, e ao desenvolvimento de um *script* que permite descarregar uma grande quantidade de vídeos do *YouTube* de forma automática.

No sistema *LiveMeans*, o *Ruby* é ainda utilizado, para efectuar a ligação entre uma página *Web* com uma *timeline* de todos os eventos ocorridos no sistema e a respectiva tabela de eventos existentes na base de dados do sistema.

3.1.2 PHP

Foi usado para uma só tarefa, aquela de, a partir de um *url* de uma página de um vídeo do *YouTube*, devolver o *url* para o *download* do seu *flv*. Este *script* é invocado pela aplicação que efectua o *download* automático de um determinado número de vídeos do *YouTube*. O facto de se usar esta tecnologia para este objectivo prende-se com o facto de, à data do seu desenvolvimento, ter sido o único exemplo que se encontrou para se conseguir obter um *url* para um vídeo do *YouTube*. Podia-se ter tentado converter o código para *Ruby*, de forma a que a aplicação não ficasse partida, mas, para além de o código se ter revelado um pouco difícil de compreender, recorrendo a funções cujo funcionamento não era claro, ainda se estavam a dar os primeiros passos na

linguagem *Ruby*. Por último, como se tratava de uma aplicação utilitária, para uso interno e sem grandes exigências não funcionais, não foi dada importância a esse *refactoring* do código.

3.1.3 C#

Foi a linguagem de programação mais utilizada para o desenvolvimento de todos os protótipos. De facto, foi em *C#* que foi desenvolvida uma aplicação de segmentação manual de vídeos, uma aplicação para criação de modelos a partir de vídeos segmentados e, acima de tudo, todo o *back-end* do *LiveMeans*. Foi ainda desenvolvido o *Atomic Remote Process Manager*, aplicação que permitia a criação de um sistema distribuído no que toca à divisão de tarefas por parte do *back-end.*

A decisão de substituir o *Ruby*, com o qual tinha sido desenvolvido o protótipo do *LiveMeans* antes do início deste projecto, pelo *C#*, envolveu uma reestruturação arquitectural da aplicação e teve como base os seguintes motivos:

- Familiaridade com a linguagem – Enquanto que os conhecimentos da linguagem *Ruby* se resumiam, a umas semanas de utilização, a situação com o *C#* era completamente diferente: Ao longo da formação académica e de outros projectos pessoais, foram desenvolvidos um grande número de aplicações com recurso ao *C#* sendo esta a linguagem com a qual se estava mais à vontade.
- Testes de *performance* – Após elaboração de um protótipo inicial do *back-end* em *C#*, fez-se o *benchmarking* para comparar as diferenças de *performance*. Nesta matéria, o *C#* saiu-se claramente melhor, chegando a ser várias vezes mais eficiente em determinadas situações.
- Legibilidade do código – Apesar dos defensores da linguagem *Ruby* argumentarem que a sintaxe da linguagem se aproxima mais da linguagem natural do que outras linguagens, a nossa conclusão foi que, na verdade, dificulta bastante a leitura do código, principalmente por pessoas alheias ao projecto. O próprio facto de se basear no modelo *Model-View-Controller(MCV)*, aumenta, na nossa opinião, a complexidade das aplicações. Consequentemente, desenvolver aplicações nesta linguagem reduz a escalabilidade e facilidade de manutenção das mesmas.
- Ausência de um *IDE* profissional – Os *IDE*'s existentes para *Ruby* são de reduzida qualidade, sendo que provavelmente a melhor opção é utilizar o *Eclipse*, que já suporta a linguagem. Contudo, o *Visual Studio 2008* possui um *IDE* extremamente robusto com um sistema de *auto-complete* muito completo, uma grande variedade de opções para *debugging*, um fácil acesso a qualquer classe, etc.

3.1.4 Java

O *Java* não foi utilizado neste projecto, mas, no entanto, foi uma possibilidade analisada. Como vantagens desta linguagem, temos o facto de ser *open source*, não necessitando de licenças para desenvolvimento, e o facto de ser portável para qualquer plataforma. Como desvantagens, salienta-se uma ligeira diminuição de *performance*, em grande parte relacionada com o arranque da máquina virtual (*Java Virtual Machine*). Destaca-se ainda o facto de a familiaridade com esta linguagem ser, apesar de tudo, menor que a existente com o *C#.*

3.1.5 Flex

O *Flex* foi utilizado por uma equipa de *design,* sub-contratada pela empresa, para desenvolver o *front-end* gráfico da aplicação *LiveMeans*. Foi necessário efectuar a integração deste *front-end* com o *back-end* desenvolvido em *C#.*

3.1.6 C++

O *C++* é a linguagem de programação na qual foram desenvolvidos os algoritmos de análise de som e de *machine learning* existentes na empresa. Apesar de estes algoritmos não terem sido alterados no decorrer deste projecto, foi necessário desenvolver um *snippet* de código em *C++* para permitir a conexão dos *audio engines*, onde estes algoritmos são executados, ao *back-end* do sistema.

3.2 Tecnologias para EAI (Enterprise Application Integration)

O *LiveMeans* necessita de tecnologias para integração das várias aplicações em que está dividido, mais concretamente, os *Audio Engines* que analisam as emissões de televisão ou rádio, o *front-end* gráfico, o *back-end*, as aplicações envolvidas no *broadcasting* das emissões em formato *flv* (*Red5*, *Vlc* e *Ffmpeg*) e o já referido *Atomic Remote Process Manager*, que permite a distribuição de tarefas do *back-end* por uma variedade de máquinas.

3.2.1 WebOrb

Na fase de desenvolvimento em que o *back-end* da aplicação ainda era o que tinha sido prototipado em *Ruby,* antes do início deste projecto, a comunicação entre este e o *front-end* gráfico era assegurada através do uso do *WebOrb*, que permite a integração do *Flex* com o *Ruby*. Esta solução revelou-se bastante pesada em termos de processamento, já que, sempre que o *front-end* necessitava de comunicar com o *back-end*, era necessário invocar um controlador do *Ruby*, sendo assim reduzida a *performance.* Para além disso, esta solução apenas permite a comunicação do cliente com o servidor, e nunca o contrário, o que obrigava a que, ao invés de o *back-end* notificar o *front-end* sempre que houvesse alguma informação nova, nomeadamente o reconhecimento de alguma entidade num determinado canal, era necessário o cliente estar constantemente a fazer *polling* ao servidor, o que resultava num grande desperdício de recursos e numa eficiência bastante baixa.

3.2.2 Web Services

Quando o *back-end* foi redesenhado em *C#*, a utilização do *WebOrb* deixou de fazer sentido e investigou-se as tecnologias que podiam ser utilizadas. A integração inicial foi efectuada recorrendo à tecnologia de *Web Services*. Esta tecnologia permite que sejam desenvolvidas aplicações para a *Web*, que consistem num conjunto de funções que, qualquer linguagem de programação que suporte *Web Services* poderá invocar, bastando apenas saber o nome das funções, o que retorna, e respectivos parâmetros de entrada. Apesar de esta tecnologia se ter revelado mais eficiente que o *WebOrb*, utilizado no protótipo anterior, continuava com um problema sério: o facto de a comunicação continuar a ser no sentido cliente-servidor e nunca servidor-cliente.

O que acontece na prática é que o *front-end* invoca uma função do *back-end* e este retorna-lhe novos dados do sistema, caso existam. O *front-end* tem de estar constantemente a efectuar este *polling* para saber se houve modificações no sistema. Uma forma de tentar contornar este problema consistiu em tentar que o cliente invocasse uma função do servidor, e este apenas retornasse qualquer informação quando existissem dados novos. Após esse retorno, o cliente invocaria novamente a função e voltaria a esperar que ela retornasse resultados. Esta solução revelou-se bastante eficiente mas foi abandonada por dois motivos: O facto de haver problemas de *time-out* e sincronismo se o servidor tardasse em retornar nova informação, e o facto de poder surgir informação nova entre o momento em que o servidor retorna novos dados e o momento em que o cliente volta a invocar a função. Era assim necessário encontrar uma solução que permitisse a fácil comunicação bidireccional entre o *front-end* e o *back-end.*

3.2.3 Sockets Assíncronas

No sentido de resolver o problema da comunicação unidireccional dos *Web Services,* foi feita uma investigação no sentido de descobrir que tecnologias poderiam ser utilizadas para permitir que a comunicação entre o *front-end* e o *back-end* fosse bidireccional. De facto, os protocolos existentes na *Web*, estão mais orientados à comunicação cliente-servidor que o contrário. Veja-se o caso, por exemplo, do famoso *AJAX*, que acenta no protocolo *HTTP*, que permite que uma página *Web* possa mudar os seus conteúdos, de acordo com dados existentes no servidor. Para tal, sempre que é necessária uma alteração, é enviado um pedido para o servidor, e este responde com os dados correspondentes. Não é possível que seja o servidor a notificar o cliente aquando da ocorrência de um determinado evento, a não ser que se utilizem tecnologias como *Comet*, que são baseadas em *polling* ao servidor, o que não é, de todo, a solução mais eficiente.

A decisão de utilizar *Sockets* tem como fundamento vários motivos:

- É das poucas soluções que existem para comunicação bidireccional entre cliente e servidor.
- Acenta sobre os protocolos de *TCP (*ou *UDP),* estando assim já implementados os mecanismos de ordenação e recuperação de pacotes de dados.
- Permite elaborar protocolos de mensagens trocadas entre cliente e servidor.
- Pode ser implementado, sem grandes dificuldades, por virtualmente qualquer linguagem de programação.
- Os dados enviados estão num buffer de bytes, o que permite o envio de informação em vários formatos, incluindo *strings* com formato *XML* para garantir uma maior escalabilidade e compatibilidade futura da aplicação.
- É extremamente rápido já que é uma implementação de baixo nível.

Por outro lado, existiam também desvantagens na utilização das mesmas, em especial a necessidade de desenvolver todo o protocolo de comunicação, algo que não seria necessário caso se tivesse optado por uma outra solução *middleware*.

Adoptaram-se as *Sockets* assíncronas, em vez de síncronas, já que aumenta, em larga escala, a eficiência do servidor, que acaba por implementar um sistema *MultiThreading* com funções de *callback,* invocadas a partir de uma nova *thread,* sempre que são recebidas novas conexões ou dados.

Este tipo de comunicação, via *Sockets* assíncronas, foi efectuado a três níveis:

- Comunicação *front-end* com o *back-end* (*Flash* / *C#*) – É a situação mais extensa, permitindo todas as funções relacionadas com o *login*, configuração e envio de alertas para o cliente.
- Comunicação *Audio-Engines* com o *back-end (C++* / *C#)* – Permite que os *Audio Engines* que estão a monitorizar as emissões de televisão/rádio, possam notificar o *back-end* sempre que sejam reconhecidas entidades nas emissões. No protótipo desenvolvido em *Ruby,* esta comunicação era bastante primitiva,

baseada na escrita das detecções, por parte dos *Audio Engines*, num ficheiro de texto, e na constante leitura desse ficheiro, de um em um segundo, por parte do *back-end.*

- Comunicação *Atomic Remote Process Managers* com o back-end (C# / *C#*) – Estes pequenos aplicativos, desenvolvidos no sentido de permitirem a distribuição de tarefas do *back-end* por diversas máquinas, correm em cada uma dessas máquinas e comunicam bidireccionalmente com o *back-end* graças ao sistema de *sockets* assíncronas.

3.2.4 Invocação de processos pelo .Net

São várias as ocasiões em que é necessário o *back-end* invocar aplicações externas, em especial no caso do *broadcasting,* em que é necessário serem lançadas e controladas várias aplicações (servidor *Red5*, *Vlc* e *Ffmpeg).* A plataforma *.Net*, na qual o *C#* acenta, permite a invocação das aplicações externas, controlar a sua execução, fecha-las e até redireccionar o *stdout* das aplicações para o próprio *back-end.*

3.3 IDEs (Integrated Development Environment) gráficos

O facto de terem sido utilizadas várias linguagens de programação no decorrer do projecto, teve como consequência a utilização de diferentes *IDE's*. Esta secção pretende efectuar uma sucinta descrição de cada um.

3.3.1 SciTE

IDE default do *Ruby*. Foi aquele que foi utilizado na fase inicial de aprendizagem da linguagem. Tinha graves limitações, tais como a ausência total de métodos de *debugging* e *auto-complete*.

3.3.2 Eclipse

Devido às limitações do *SciTe*, este foi o *IDE* utilizado aquando do desenvolvimento do script que efectua o *Web Crawling* do *YouTube* para descarregar grandes quantidades de vídeos.

3.3.3 Visual Studio 2008

Após o abandono do *Ruby* como linguagem de programação para desenvolvimento e adopção do *C#*, o *Visual Studio 2008*, recentemente lançado, revelou-se o *IDE* mais avançado e completo com que se trabalhou, não só com *C#,* mas com qualquer linguagem de programação. Entre outras funcionalidades, o encapsulamento automático de código, a geração de métodos, as sugestões de correcção de erros, o *auto-complete* bastante avançado e intuitivo, o imenso potencial das funções de *debugging*, reforçaram a opinião que este é o *IDE* mais avançado do momento no que toca à usabilidade e produtividade. Na Figura 3.1, é possível visualizar-se uma janela de funcionamento do *Visual Studio 2008* aquando do desenvolvimento do *back-end* do *LiveMeans*.

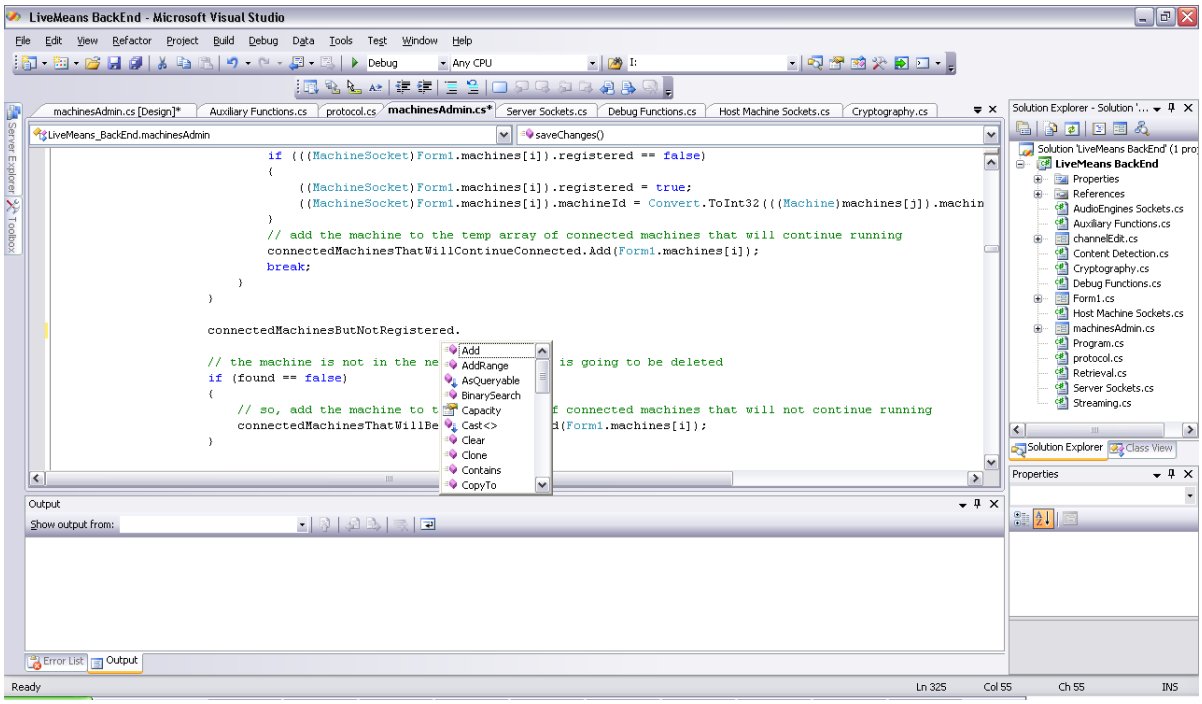

Figura 3.1 – Desenvolvimento do *Back-End* do *LiveMeans* no *Visual Studio 2008*

3.4 Formato da informação enviada entre aplicações

Sempre que são utilizadas tecnologias para *EAI (Enterprise Application Integration*), em especial na utilização de *Sockets,* o formato com que as mensagens são trocadas entre as aplicações (*Audio Engines*, *front-ends, atomic remote process managers* e *back-end)* teve de ser analisado.

3.4.1 CSV (Comma Separated Values)

Inicialmente, no sentido de encurtar ao máximo o tamanho das mensagens trocadas, enviava-se as mensagens separando-se os diferentes campos por vírgulas ou outros caracteres delimitadores. Este método tinha como grande vantagem a sua velocidade, já que o tamanho das mensagens era bastante curto.

3.4.2 XML (Extensible Markup Language)

Apesar do formato *CSV* permitir um menor número de *bytes* trocados entre as aplicações, a realidade é que as mensagens trocadas nunca eram muito extensas, e chegou-se à conclusão que, a adopção do formato *XML*, apesar de praticamente duplicar a quantidade de informação trocada, permite uma muito maior facilidade de interpretação dos dados (*parsing*), aumento de compatibilidade com outras aplicações e consequente aumento de escalabilidade e facilidade de manutenção futura.

3.5 Sistema de Gestão de Bases de Dados

Nesta secção, irão ser descritas as tecnologias utilizadas para gestão de bases de dados, quer o *SGBD*, quer o cliente utilizado para interagir com este.

3.5.1 MySQL

O *MySQL* foi o *SGBD* (Sistema de Gestão de Bases de Dados) escolhido. Esta decisão prende-se, acima de tudo com a familiaridade com esta tecnologia, a boa aceitação da mesma na comunidade informática e o facto de todas as bases de dados da empresa se basearem nesta tecnologia.

3.5.2 AB MySQL Administrator

Ao longo da formação académica, muitos foram os clientes *MySql* utilizados, tais como o *SqlYog, EMS SQL* ou o *DreamCoder*. Muitos deles revelaram-se insuficientes, quer pela existência de *bugs*, quer pela fraca usabilidade ou robustez. O *AB MySQL Administrator*, para além de ter sido desenvolvido pelos criadores do *MySQL,* revelou-se um cliente extremamente robusto e eficaz.

3.6 Tecnologias para BroadCasting

Uma das características do *LiveMeans* é a transmissão *live* de emissões televisivas. Para tal, é necessário utilizar o *VLC Media Player* para capturar a *stream*, converter num formato adequado e enviar para um porto local. De seguida, é necessário utilizar o *Ffmpeg* para capturar essa *stream*, converter no formato *Flash Player*, e enviar para o servidor *RTMP* do *Red5,* que por sua vez trata de transmitir a emissão para todos os clientes da aplicação.

3.6.1 Red5

O *Red5* é um servidor *RTMP (Real Time Messaging Protocol) Open Source* programado em *Java*. Para além de permitir o *liveStream* de vídeos locais para *Flash Players* remotos, tem como grande vantagem o facto de também permitir efectuar a transmissão de conteúdos que já são *liveStream*, ou seja, com a ajuda do *VLC e Ffmpeg*, capturar uma emissão *live* e retransmiti-la em formato *FLV (Flash Video)*.

As modificações efectuadas ao *Red5,* por parte da empresa, de forma a permitir o *multiStream,* ou seja, a emissão de mais do que um conteúdo *live* ao mesmo tempo, tornaram esta tecnologia mais interessante que outras não livres, tal como o *Adobe Media Center*. Sem estas modificações, a arquitectura do *LiveMeans* seria bastante mais complexa, já que exigiria que existisse um servidor *Red5* (em cada máquina) por cada canal de televisão que se estivesse a retransmitir para o *flash player* do *front-end*. Assim, o número de emissões *live* que cada servidor *Red5* pode transmitir é apenas limitado pela capacidade de processamento da máquina em que se encontra.

3.6.2 Ffmpeg

O *Ffmpeg* é um programa que permite gravar, converter e efectuar o *stream* de sinal áudio e vídeo digital em diversos formatos. É uma ferramenta de linha de comandos constituída por uma colecção de bibliotecas *open source*. É com esta ferramenta que, no sistema *LiveMeans*, os formatos em que outras emissões são transmitidos, são convertidos para *Flash Video (FLV)* e enviados de seguida para o servidor *RTMP* do *Red5*. Infelizmente, o *Ffmpeg* não permite capturar um *stream* remoto pelo que é necessário utilizar o *VLC* como intermediário.

3.6.3 VLC Media Player

O *VLC Media Player* é um *media player* aberto e livre. Permite, para além da reprodução de conteúdos audiovisuais, a codificação, conversão e *streaming,* tendo embebidos uma grande quantidade de *codecs* e protocolos de *streaming.* No sistema

LiveMeans, é utilizado para capturar um *stream* de uma emissão *live*, converter para um formato adequado aos algoritmos de análise de sinal (tal como o ajuste do *bit rate* para 22050) e o envio para um porto local, futuramente capturado pelo *ffmpeg,* que o converterá para *Flash Video* e enviará para o servidor *RTMP* do *Red5*, responsável então pelo *broadcast* da emissão para todos os clientes da aplicação.

3.6.4 Flash Video

O *Flash Video*, conhecido simplesmente como *FLV,* é um formato de ficheiro utilizado para transmitir vídeo pela *Internet*, através da tecnologia *Flash* da *Adobe.* Este formato tem grandes vantagens, tais como a maior compatibilidade, o encapsulamento do vídeo, maior protecção e facilidade de uso, tendo já como utilizadores massivos, o *YouTube*, o *Google Video*, o *Reuters.com*, ou o *Yahoo! Video*. Existem inclusive, vários canais televisivos que disponibilizam os seus conteúdos *online*, neste formato.

3.7 Tecnologias para alerta remoto

Uma das características interessantes do *LiveMeans* é o facto de, caso um utilizador não esteja ligado ao sistema no momento de detecção de uma determinada entidade, é, ainda assim notificado, via *e-mail* ou *sms*. Assim, se por exemplo, um utilizador estiver fora de casa no momento em que um político, no qual ele está interessado, começar a falar na Assembleia da República, é notificado via *sms* de que esse político foi reconhecido num determinado canal, podendo então assistir à emissão, ou, caso não tenha disponibilidade, esta é gravada para posterior visualização.

3.7.1 SMS Gateway

Para a entrega de *sms's* de notificação, foi utilizada a *Gateway* de *sms* da *Shortcut* [SHO08], uma empresa portuguesa com clientes como a *Tmn* e a *PréNatal*.

3.7.2 Google SMTP (Simple Mail Transfer Protocol)

Para o envio dos *e-mails* de notificação, foi criada uma conta no *Gmail* (clustermedia@gmail.com), enviando-se então os e-mails a partir desta conta, bastando para isso aceder ao servidor de *SMTP* do Google.

3.8 Sistemas de gestão de versões

Nesta secção, irá ser descrita a tecnologia utilizada para gestão de versões, assim como o cliente utilizado para interagir com esta.

3.8.1 SVN

É o sistema de gestão de versões mais popular, e o facto de a empresa já o ter completamente integrado em todos os projectos, facilitou a decisão de adoptar este sistema para gerir o projecto. As vantagens em utilizar um sistema destes são óbvias, tais como uma maior segurança em caso de perda de dados ou de alterações acidentais da aplicação, ou a possibilidade de se visualizar as alterações entre as várias versões desenvolvidas. No caso do *LiveMeans*, a máquina de desenvolvimento variava consoante se estivesse a trabalhar na máquina de desenvolvimento, no *laptop* pessoal, ou apenas a fazer pequenas modificações na versão a correr no servidor. Assim, a utilização do *SVN* permitiu que nunca houvesse problemas e garantiu que se trabalhava sempre com a versão mais recente, independentemente da máquina que se utilizava.

3.8.2 TortoiseSVN

O *TortoiseSVN* era o cliente de *SVN* adoptado pela empresa, o que, a aliar ao facto de ser aquele com que se tinha mais experiência, e também um dos mais aceites pela comunidade informática, levou à decisão de o utilizar para interagir com o servidor de *SVN* existente na empresa.

3.9 Servidores Web

Para o funcionamento de todo o sistema *LiveMeans* e respectivas aplicações adicionais, foi necessário a implementação de dois *Web Servers*, o *Apache* e o *Microsoft IIS*.

3.9.1 Apache

O servidor *Apache* é utilizado para um único propósito: Permitir a utilização do *script*, desenvolvido em *PHP*, que, a partir do *url* de uma página de um vídeo do *YouTube*, obtém o *url* directo para o *flv* deste. Esta função é utilizada pelo *YouTube Flvs Crawler*, aplicação desenvolvida que permite o descarregamento massivo de vídeos do *YouTube.*

3.9.2 IIS

O *Web Server IIS* é utilizado para um caso de uso do *LiveMeans*: O *Upload* de um vídeo de um utilizador. De facto, para se conseguir enviar um vídeo para o servidor, a partir de um cliente *Flash*, é necessário a utilização de um *script server-side*. Poder-se-ia ter utilizado novamente o conjunto *PHP/Apache* mas, na altura em que esta funcionalidade estava a ser desenvolvida, existia a possibilidade de tornar todo o *back-end* do *LiveMeans* uma *Web Application* e, assim sendo, a utilização do *IIS* poderia eventualmente permitir a fusão deste *script* com o restante *back-end*.

3.10 Tecnologias de Encriptação de dados

A questão da segurança é cada vez mais válida, tendo-lhe sido atribuída uma grande importância no desenvolvimento do *LiveMeans*, em especial no caso de autenticação de utilizadores. Irão ser descritas as tecnologias utilizadas, sendo que, para pormenores de implementação, é sugerida a análise do capítulo 6.3.

3.10.1 RSA

Este algoritmo é utilizado para encriptar as palavras-chave na base de dados. É utilizado porque permite a recuperação de dados a partir da chave de desencriptação única existente no servidor.

3.10.2 MD5

O *MD5* é um algoritmo de encriptação de dados com perda de informação e, consequentemente, não recuperável. É utilizado no envio das palavras-chave do cliente para o servidor, de forma a não expor essa informação. Para validação, o servidor pode comparar as *hashes* resultantes da encriptação com este algorítmo e verificar se coincidem.

Capítulo 4

Solução Proposta

Neste capítulo irá ser descrita a proposta para resolver os problemas existentes e criar um produto de valor, inserido no contexto da empresa. Para tal, vão ser apresentadas todas as aplicações desenvolvidas, que juntas formam todo o sistema *LiveMeans*. Será ainda apresentada, de forma conceptual e não mapeada a tecnologias, uma arquitectura de todo o projecto. Para cada aplicação desenvolvida, serão enunciados os casos de uso que representam os requisitos pretendidos.

4.1 Arquitectura

Como é possível observar no diagrama presente na Figura 4.1, a solução baseia-se no modelo cliente-servidor. Um utilizador da aplicação, representado a azul, acede a ela através do cliente *web*, utilizando um *web browser,* que se conecta ao servidor.

O servidor, peça central deste sistema, responsável por várias tarefas, encontra-se dividido por várias máquinas:

- Máquina de *Back-End* – É a esta máquina que os clientes se conectam e interagem com o servidor.
- Servidor da base de dados – Contém a base de dados da aplicação.
- Máquinas de *Streaming* – Máquinas que são responsáveis pela retransmissão dos canais televisivos.
- Máquinas de Análise – Máquinas responsáveis pela monitorização e reconhecimento de entidades nos canais audiovisuais.

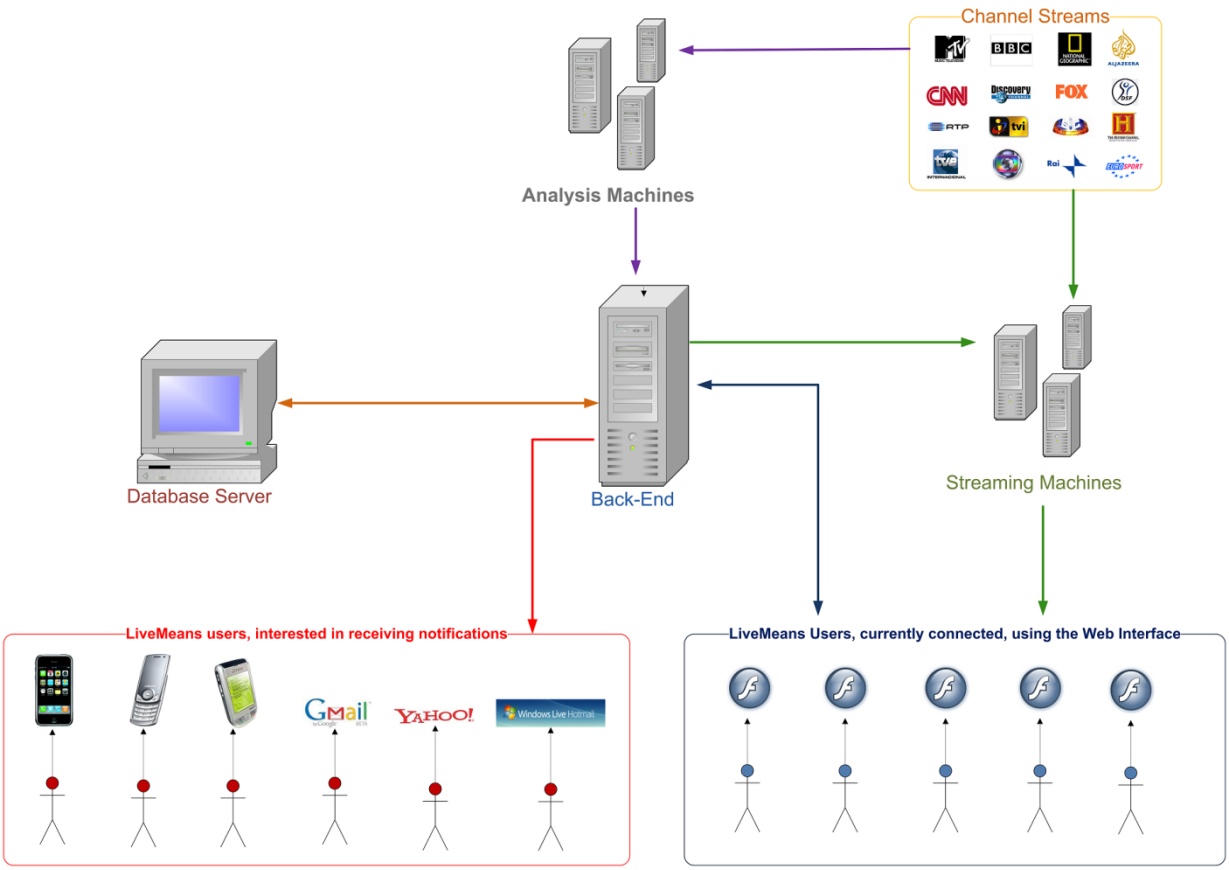

Figura 4.1 – Arquitectura Proposta para o sistema *LiveMeans*

As máquinas de análise, sendo aquelas que dão sentido a todo o sistema, utilizam os modelos de som e imagem criados por aplicações de apoio (*YouTube Flvs Crawler*, Segmentador de vídeos, *Model Creator* e *Model Tester*), que irão ser descritas nas próximas secções. Estas aplicações têm como objectivos a obtenção de uma colecção de conteúdos audiovisuais e a criação de modelos de som e imagem para utilização pelas máquinas de análise no reconhecimento de entidades.

A arquitectura proposta, contempla ainda um módulo de notificação remota, utilizado para notificar os utilizadores remotamente, via *sms* e/ou *e-mail.* No que toca à base de dados, esta será utilizada para armazenar todo o tipo de dados do sistema, desde informação dos utilizadores e interesses de cada um nas respectivas entidades, às informações sobre os canais *live* e as máquinas responsáveis por os transmitir.

4.2 Aplicações de apoio ao LiveMeans

Nesta secção irão ser descritas as aplicações desenvolvidas que, não pertencendo directamente ao sistema final do *LiveMeans*, partilham a sua base de dados e possibilitam a existência de uma colecção de conteúdos audiovisuais e a criação de

modelos de reconhecimento de som e imagem. Estes modelos são a base das máquinas de análise de conteúdos, que dão sentido a todo o sistema *LiveMeans*.

4.2.1 YouTube Flvs Crawler

Esta aplicação tem como objectivo reunir uma colecção de vídeos de determinadas entidades. Assim, possibilita que, mediante a inserção de palavras-chave que correspondam à entidade que se procura, sejam retirados do servidor do *YouTube* um determinado número de vídeos relacionados com essa entidade. Permite ainda definir o número de vídeos que se pretende descarregar. Na Figura 4.2, podemos analisar um exemplo de utilização da aplicação, em que se pretende extraír do servidor do *YouTube*, 400 vídeos que contenham a tag *Barack Obama*.

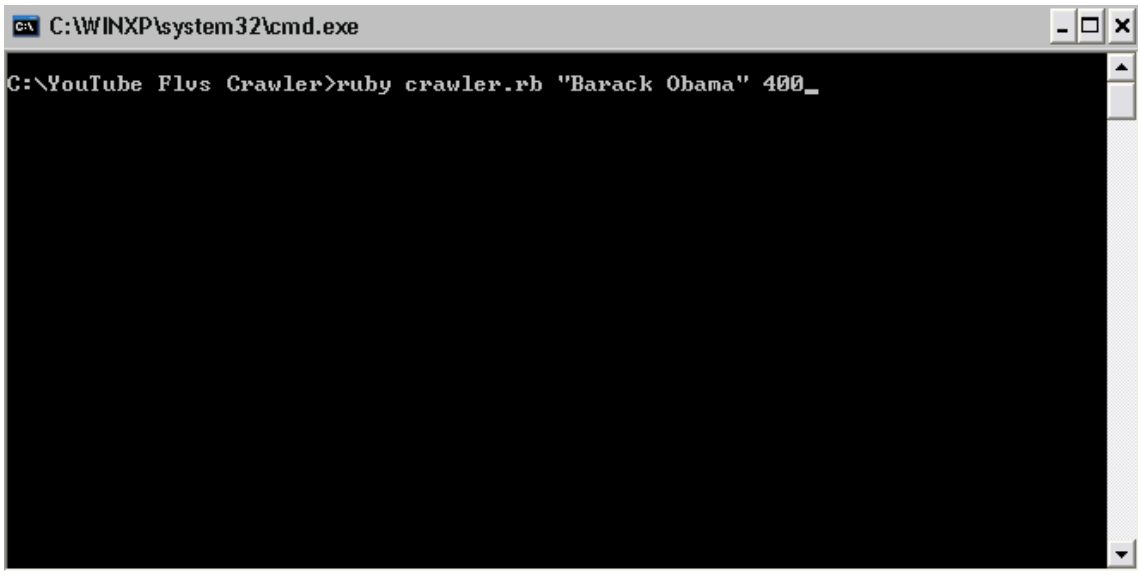

Figura 4.2 – Exemplo de utilização do *YouTube* Flvs Crawler

Posteriormente, estes vídeos poderão ser segmentados de forma manual (*ground truth*), sendo para esse efeito utilizada a aplicação descrita na próxima secção.

4.2.2 Segmentador de vídeos

Esta aplicação gráfica permite visualizar e segmentar por *tags,* os vídeos existentes numa base de dados de conteúdos audiovisuais. Esta base de dados pode ter sido preenchida através da utilização do *YouTube Flvs Crawler*, de uma outra aplicação ou até de forma manual.

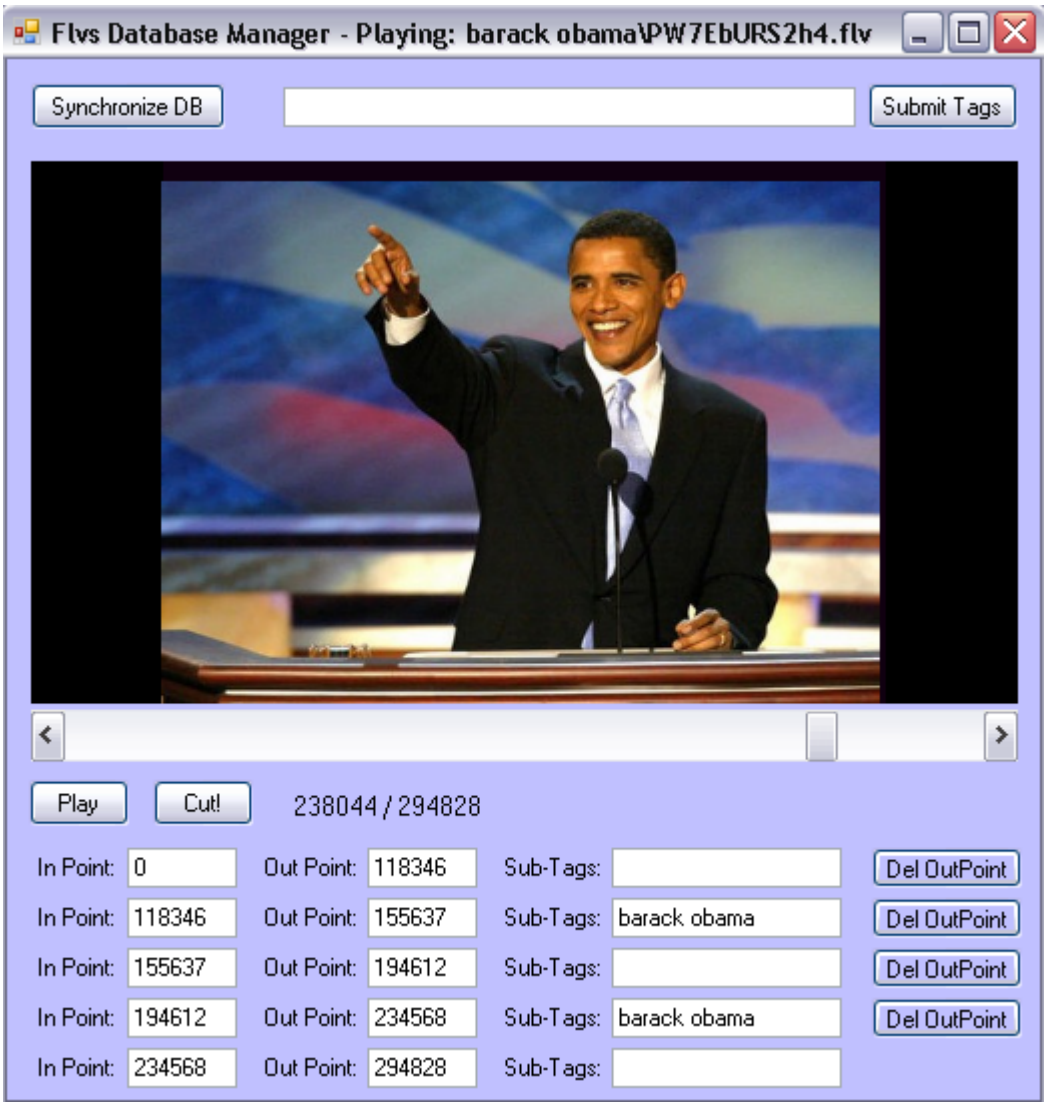

Figura 4.3 – Utilização do Segmentador de Vídeos para definir regiões de interesse num vídeo

É assim possível definir regiões, com um *inPoint* e *outPoint,* e associar *tags* a essas regiões. Esta aplicação permite assim efectuar a segmentação manual de vídeos existentes no repositório, ou seja, o chamado *ground truth*. Na Figura 4.3 podemos analisar um exemplo de utilização da ferramenta. Podemos verificar que foram criados *keyPoints*, com o botão *Cut*, nos pontos *118.346, 155.637, 194.612* e *294.828 (s)*. Desta forma, foram criadas as cinco regiões que são delimitadas por esses pontos, tendo duas delas sido segmentadas com a *tag Barack Obama.* Esta segmentação irá posteriormente ser utilizada pelo *model creator* para criação dos modelos de som e imagem.

Na Figura 4.4 podemos analisar o diagrama de casos de uso desta ferramenta.

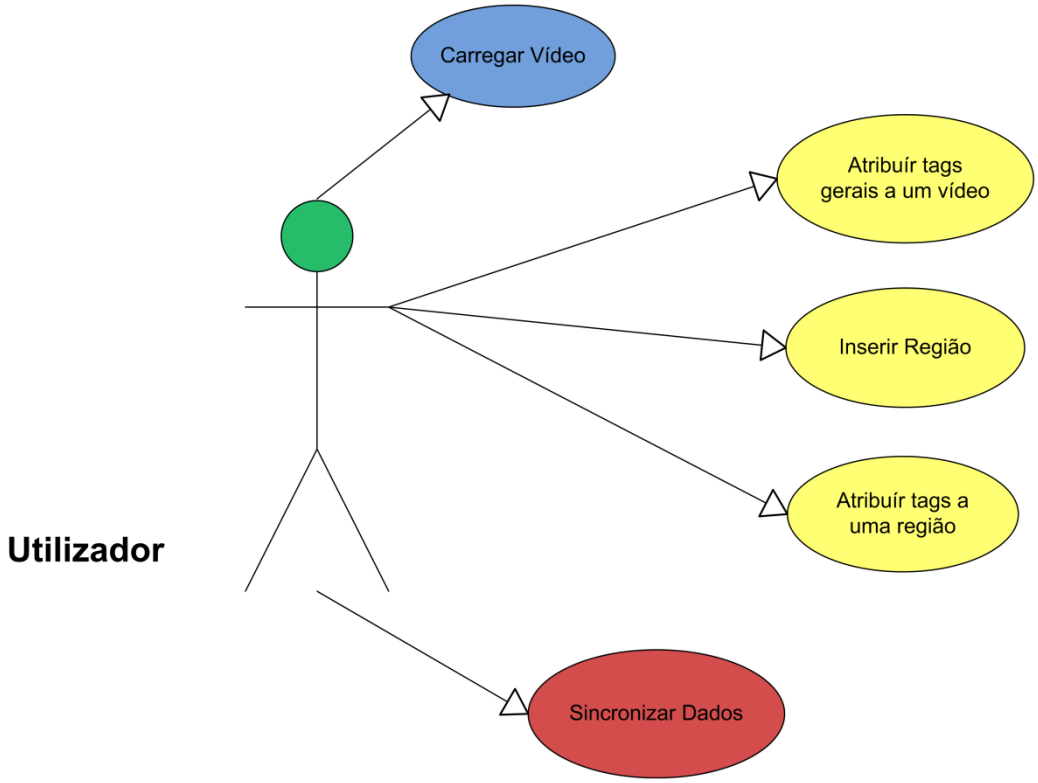

Figura 4.4 – Diagrama de Casos de Uso do Segmentador de Vídeos

Estes casos de uso são de seguida descritos:

- **Carregar Vídeo** – O sistema carrega um vídeo que ainda não tenha sido segmentado, para que o utilizador o segmente.
- **Atribuir uma *tag* ao vídeo** – É possível, na caixa de texto no topo da aplicação, atribuir uma *tag* ao vídeo no seu geral, e não apenas a uma determinada região do mesmo.
- **Inserir Região** – Ao inserir um *keyPoint*, o sistema automaticamente divide em dois a região onde esse *keyPoint* foi inserido, sendo acrescentada à *form*, uma nova região, ou seja, um novo conjunto de controlos que permitem a inserção de uma *tag* que a caracterize. É ainda possível eliminar essa região através de um botão.
- **Sincronizar dados** – O sistema permite a sincronização de dados entre a pasta física que actua como repositório dos vídeos e a base de dados. Assim, são eliminados da base de dados vídeos que já não existam fisicamente, e são adicionados aqueles que não estão presentes na base de dados mas que existem fisicamente.

4.2.3 Model Creator & Model Tester

O *Model Creator* e o *Model Tester* são dois módulos da mesma aplicação. O primeiro permite, a partir da segmentação efectuada com o segmentador de vídeos, criar modelos para reconhecimento de entidades. Permite que o utilizador defina quais são as entidades para as quais pretende criar os modelos, e o sistema encarrega-se de analisar todos os excertos de vídeos que tenham sido segmentados com *tags* correspondente a essas entidades, trata o áudio para, com os algoritmos da empresa, criar modelos de som de cada entidade, e, caso as entidades sejam pessoas, extrair *frames* do vídeo para, com as bibliotecas alteradas de reconhecimento de faces, criar modelos da face de cada orador.

Com o *Model Tester,* o sistema completa-se, já que este é responsável pela análise de um novo ficheiro audiovisual, permitindo, através do recurso aos modelos de som e faces criados no *Model Creator*, reconhecer os oradores presentes neste novo conteúdo.

4.3 LiveMeans

As outras aplicações descritas neste capítulo, apesar de não pertencerem directamente ao sistema *LiveMeans*, permitem a sua existência, já que em conjunto permitem a criação de uma base de dados de conteúdos audiovisuais, segmentadas por entidades de forma manual (*Ground-Truth*) e, essencialmente, a criação de modelos de som e de imagem que são a base em que todo o sistema acenta, e aquela que o torna realmente útil e inovador.

A solução apresentada, designada, como já foi referido, por *LiveMeans*, pretende resolver os problemas existentes na indústria de conteúdos audiovisuais que foram referidas no segundo capítulo e cumprir os objectivos pretendidos aquando do desenvolvimento do projecto.

Assim sendo, desenvolveu-se uma aplicação que acenta no modelo cliente-servidor, em que o servidor é responsável pela monitorização e análise de canais rádio-televisivos, no sentido de, utilizando os algoritmos e modelos da empresa, reconhecer as entidades presentes nessas emissões. Já o cliente consiste numa aplicação *Web*, acedida por um *WebBrowser*, que permite que os utilizadores visualizem as emissões e sejam notificados dos conteúdos reconhecidos. Os utilizadores podem ainda submeter vídeos seus para análise e partilhá-los com outros utilizadores interessados nos conteúdos detectados. De notar ainda a possibilidade de os utilizadores, se não estiverem conectados à aplicação, poderem ainda assim ser notificados via *sms* ou *e-mail*.

4.3.1 Cliente

O cliente do *LiveMeans* é uma aplicação Web, acedida a partir de um *web browser*, utilizada pelo utilizador final de forma a interagir com o sistema, no sentido de visualizar as emissões, adicionar e remover alertas, enviar novos víeos, etc. Na Figura 4.5 é possivel visualizar-se um exemplo de utilização da aplicação. Neste caso, verificamos que existe um *player* central a reproduzir um determinado canal, e que na secção inferior existem *thumbnails* de todos os canais existentes, sendo cada um desses *thumbnails* um *mini-player* que permite a pre-visualização dos canais.

Sempre que é reconhecida uma determinada entidade, na qual o utilizador está interessado, surge uma notificação gráfica e sonora por cima do canal correspondente, para além da eventual notificação remota via *sms* ou *e-mail*. De notar ainda que nos *thumbnails* presentes na Figura 4.5, apenas se encontram emissões *live*. Caso sejam reconhecidas, num vídeo enviado por um outro utilizador, entidades nas quais o utilizador está interessado, surge um novo *thumbnail,* com o vídeo que foi transferido, possibilitando a sua visualização no *player* central. Esta funcionalidade é designada por *Video On Demand*.

Figura 4.5 – Janela principal do cliente *LiveMeans* com a reprodução de uma emissão *live*

Para além da janela principal, o utilizador pode aceder a uma janela de *settings*, na qual define quais as entidades em que está interessado, através de um sistema intuitivo de *drag and drop.* Esta janela pode ser analizada na Figura 4.6.

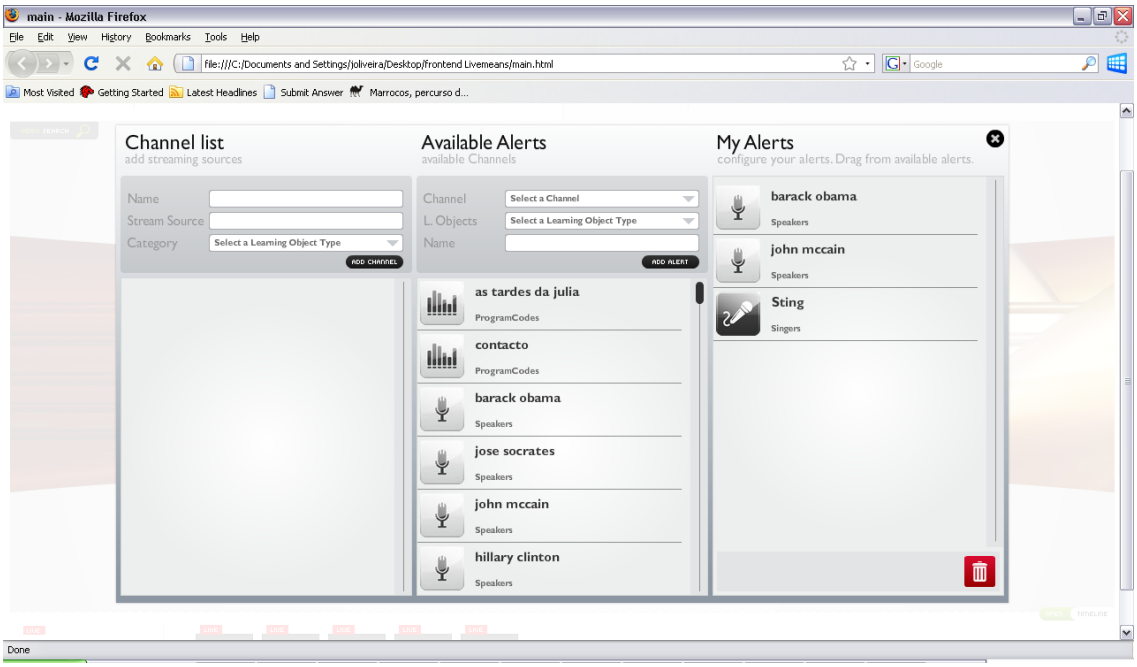

Figura 4.6 – Janela de definições do cliente *LiveMeans*

A Figura 4.7 traduz os casos de uso deste módulo do sistema.

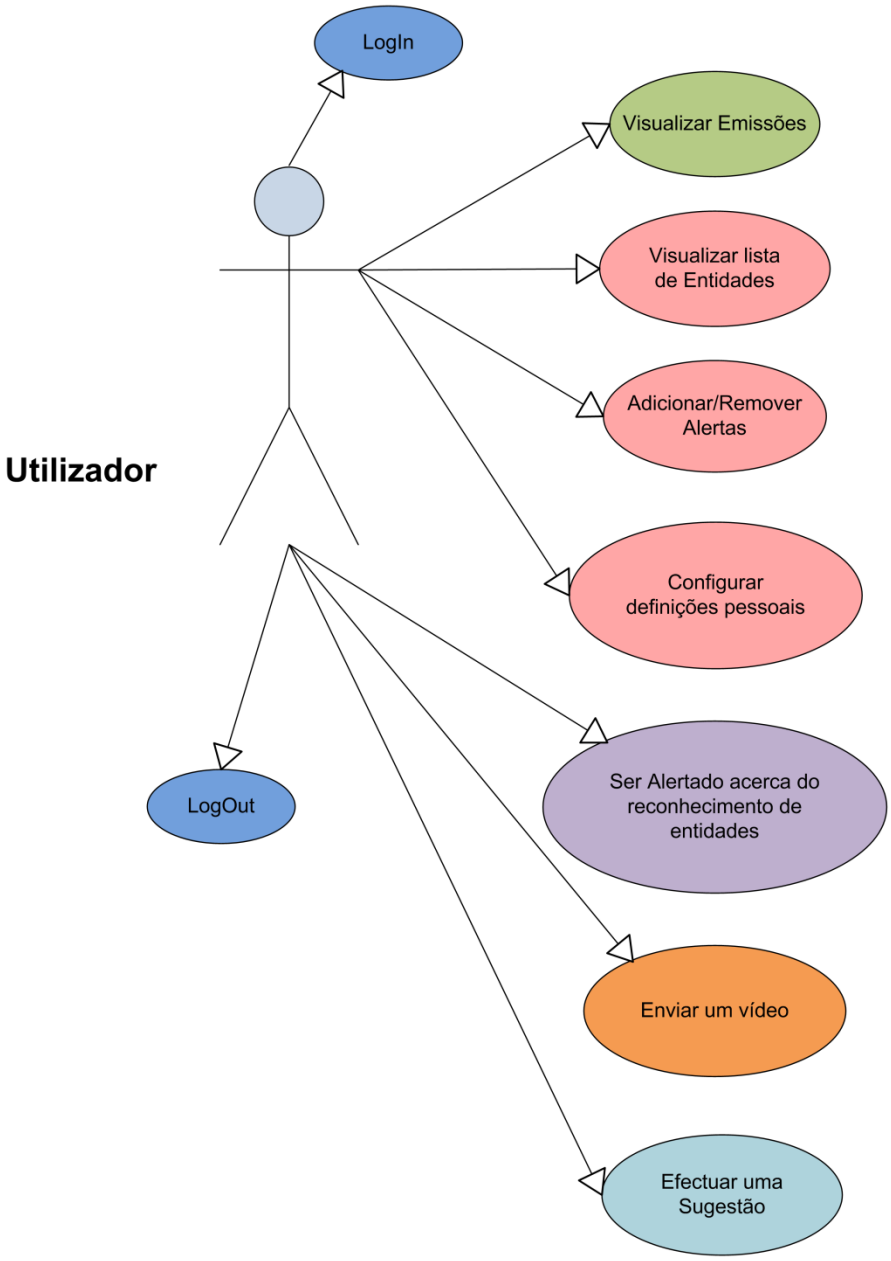

Figura 4.7 – Diagrama de Casos de Uso do Cliente *LiveMeans*

Esses casos de uso são descritos de seguida:

- ***Login*** – Permite que um determinado utilizador se autentique na aplicação, inserindo o seu nome de utilizador e palavra-chave. Após um *login* válido, o utilizador tem então acesso a todas as funcionalidades da aplicação. Foi dada bastante importância à questão de autenticação, tendo sido criado um sistema complexo e cujos detalhes de implementação podem ser analisados no capítulo 6.3.
- **Visualização das emissões *live*** – A janela principal da aplicação permite que os utilizadores, sem terem configurado qualquer coisa, visualizarem já todas as emissões televisivas que estão a ser enviadas pelo *back-end* em pequenos

thumbnails na parte inferior do ecrã. É então possível arrastar um desses *thumbnails* para o *player* central e visualizar a emissão no ecrã principal.

- **Visualizar a lista de alertas disponíveis** – Na janela de configuração, o utilizador pode visualizar todos os alertas disponíveis, ou seja, todas as entidades que o sistema está preparado para reconhecer, que podem ser de diversos tipos, tais como oradores, programas, publicidade, música, linguagens, etc.
- **Adicionar alertas** – Um utilizador pode adicionar um alerta, de forma a poder ser notificado sempre que esse alerta é reconhecido, quer nos conteúdos *live*, quer nos conteúdos detectados em vídeos enviados por outros utilizadores.
- **Remover alertas** – É possível remover alertas que tenham sido adicionados previamente.
- **Configurar definições pessoais** – O utilizador pode alterar os parâmetros do seu perfil. Pode, por exemplo, definir a forma de notificação remota ou os seus dados pessoais.
- **Enviar um vídeo seu para o servidor** – Um utilizador pode efectuar o *upload* de um ficheiro pessoal para o servidor para que este analise os seus conteúdos semânticos. Este módulo *Video On Demand* corresponde a uma adaptação do *Model Tester* descrito no capítulo 4.2.3.
- **Ser notificado do reconhecimento de uma entidade** – Este caso de uso não é despoletado pelo utilizador mas sim pelo servidor, sempre que detecta um conteúdo. Assim, um utilizador interessado numa determinada entidade é notificado sempre que esta seja reconhecida. A notificação, caso o utilizador esteja a usar a aplicação, é feita pelo cliente *Web*. Caso não o esteja, pode ser notificado remotamente, de acordo com as suas preferências, por *sms* ou *e-mail*.
- **Efectuar uma sugestão** – O utilizador pode sugerir que se criem modelos para determinadas entidades, ainda não existentes no sistema. Esta funcionalidade permite obter *feedback* dos utilizadores e perceber aquilo em que eles estão interessados.
- **LogOut** – Abandonar a aplicação.

4.3.2 Back-End

O *back-end* do Livemeans possui uma interface gráfica que permite aos administradores do sistema efectuar, entre outras tarefas secundárias, a gestão de máquinas, emissões e utilizadores.

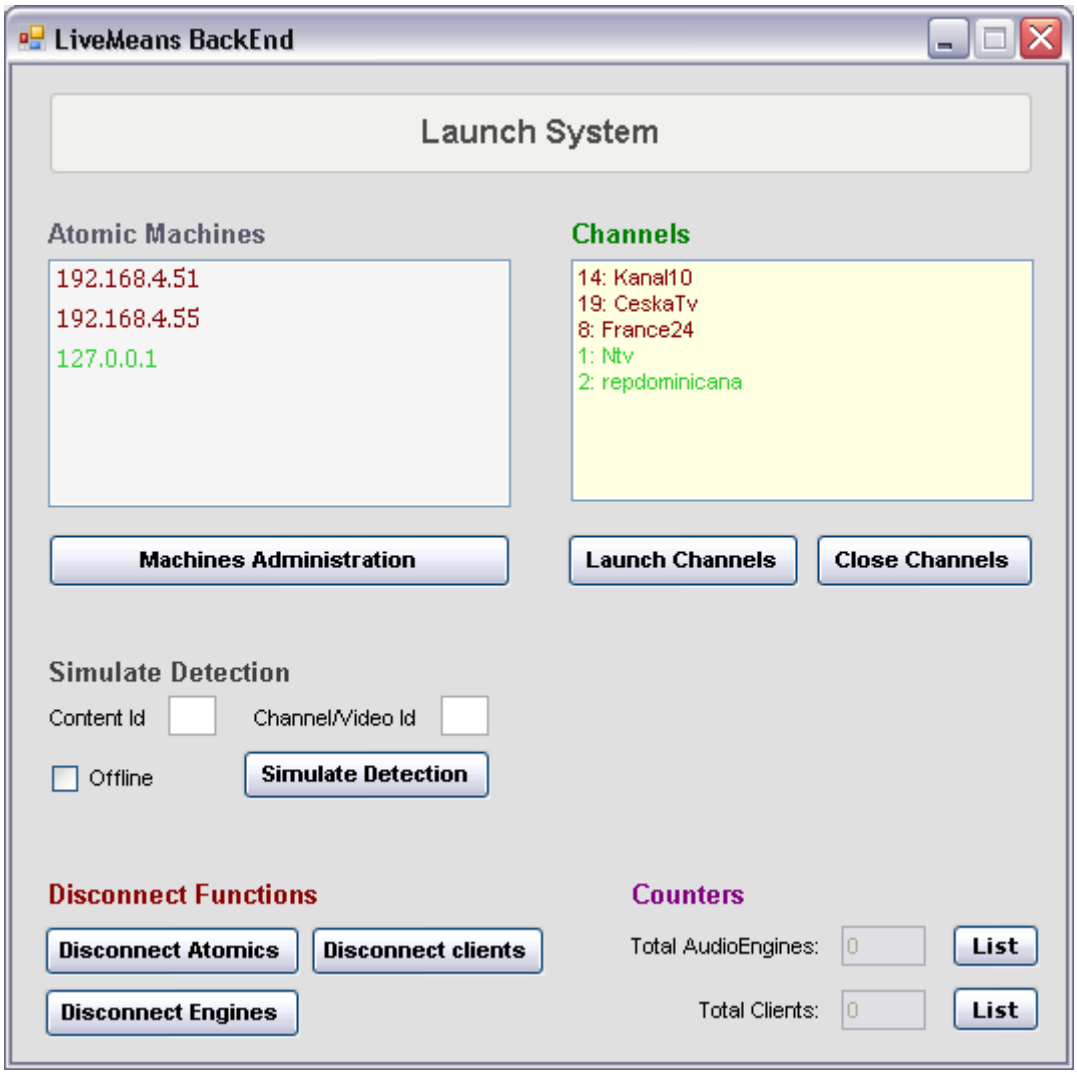

Figura 4.8 – *Back-End LiveMeans* com uma máquina conectada e dois canais em emissão

A *interface* gráfica do *back-end* encontra-se dividida em duas *forms* diferentes:

1. Uma para, entre outras possibilidades, se visualizarem os utilizadores e máquinas remotas conectadas, iniciar e terminar emissões de canais televivisos, e simular detecções.
2. Uma outra *form* com uma interface dinâmica e baseada em *drag and drop*, para administração do sistema distribuído de emissões, sendo possível, apenas com o uso do rato, a distribuição dos vários canais pelas máquinas de *streaming* existentes, ou seja, definir que máquinas vão ser responsáveis pela emissão de quais canais. Esta *form* permite ainda a criação, edição e remoção dinâmica de máquinas de *streaming* e de canais televisivos. Existe ainda uma *trashbin*, designada *Not Allocated Channels Container*, onde são colocados os canais que não vão ser transmitidos por nenhuma máquina, ou seja, um *container* para os canais que não se pretende eliminar mas também, de momento, não se pretende transmitir.

Os detalhes de implementação destas duas *forms* gráficas são descritos no capítulo 6.7.

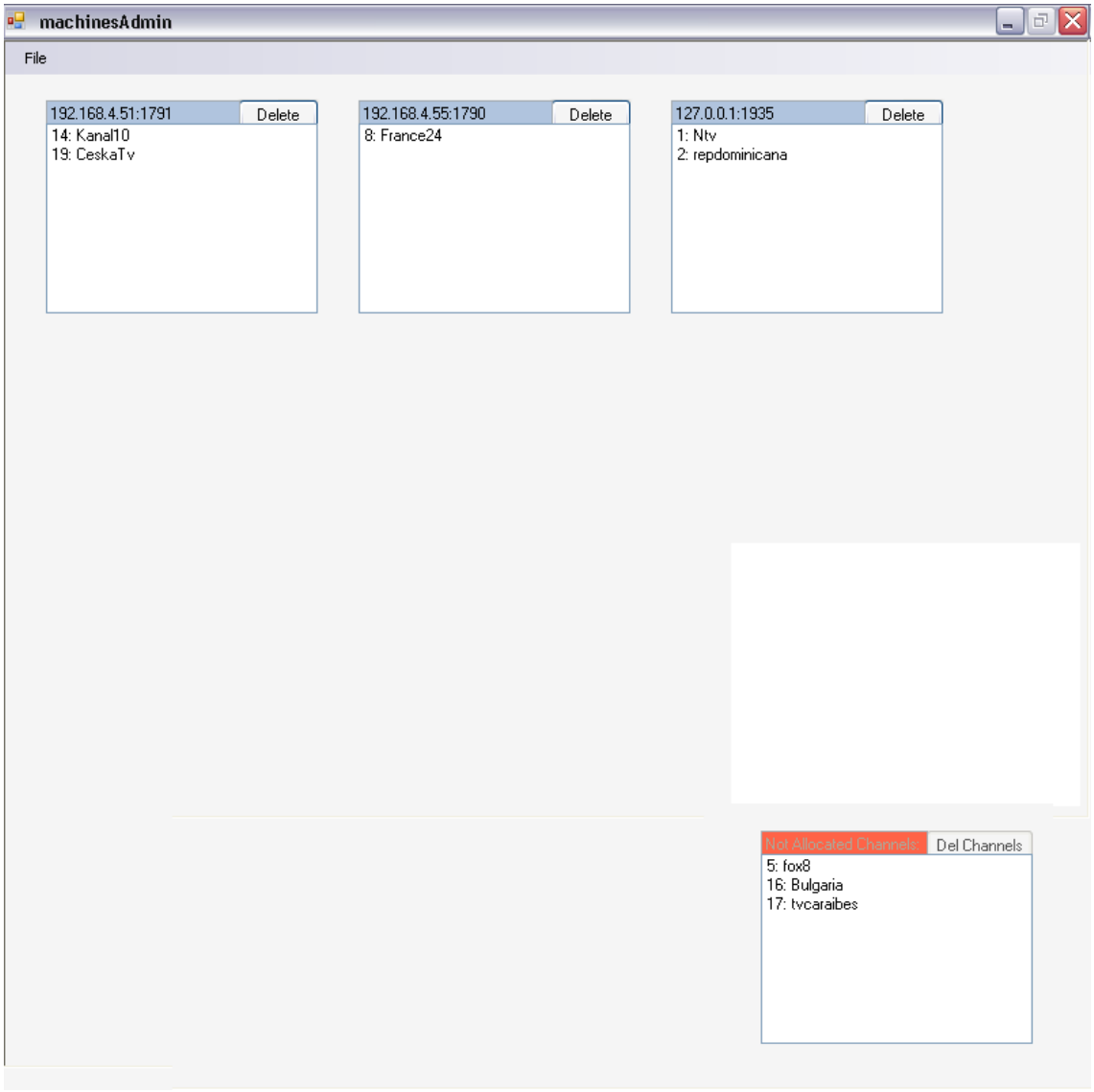

Figura 4.9 – Janela de administração de máquinas e canais do *back-end* do *LiveMeans*

Na Figura 4.8 pode-se visualizar um exemplo de utilização da *form* principal do *back-end*, em que podemos verificar que não se encontra conectado nenhum utilizador ou máquina de análise. Para além disso, verificamos que, das três máquinas de *broadcasting* registadas na base de dados, apenas uma delas se encontra conectada, sendo tal estado representado pela cor verde. Podemos ainda ver que, de todos os registos de canais existentes, apenas dois se encontram ligados. Estes dois canais têm de estar forçosamente a ser transmitidos na máquina *127.0.0.1* já que é a única que se encontra ligada. De notar que o utilizador pode selecionar uma ou várias máquinas, sendo apenas apresentados os canais pelos quais essas máquinas estão responsáveis.

Na Figura 4.9, é exemplificada a segunda *interface* do *back-end*, responsável pela gestão de máquinas e distribuição de canais. Neste exemplo, coerente com a Figura

4.8, podemos verificar que existem três máquinas, cada uma com um ou dois canais. Verifica-se ainda que existem três canais desactivados, ou seja, sem nenhuma máquina que os transmita, estando portanto colocados no *container* existente para o efeito, no canto inferior direito. É possível o arrastar de qualquer canal de um *container* para outro.

Os casos de uso do *back-end* do *LiveMeans* são enunciados no diagrama presente na Figura 4.10.

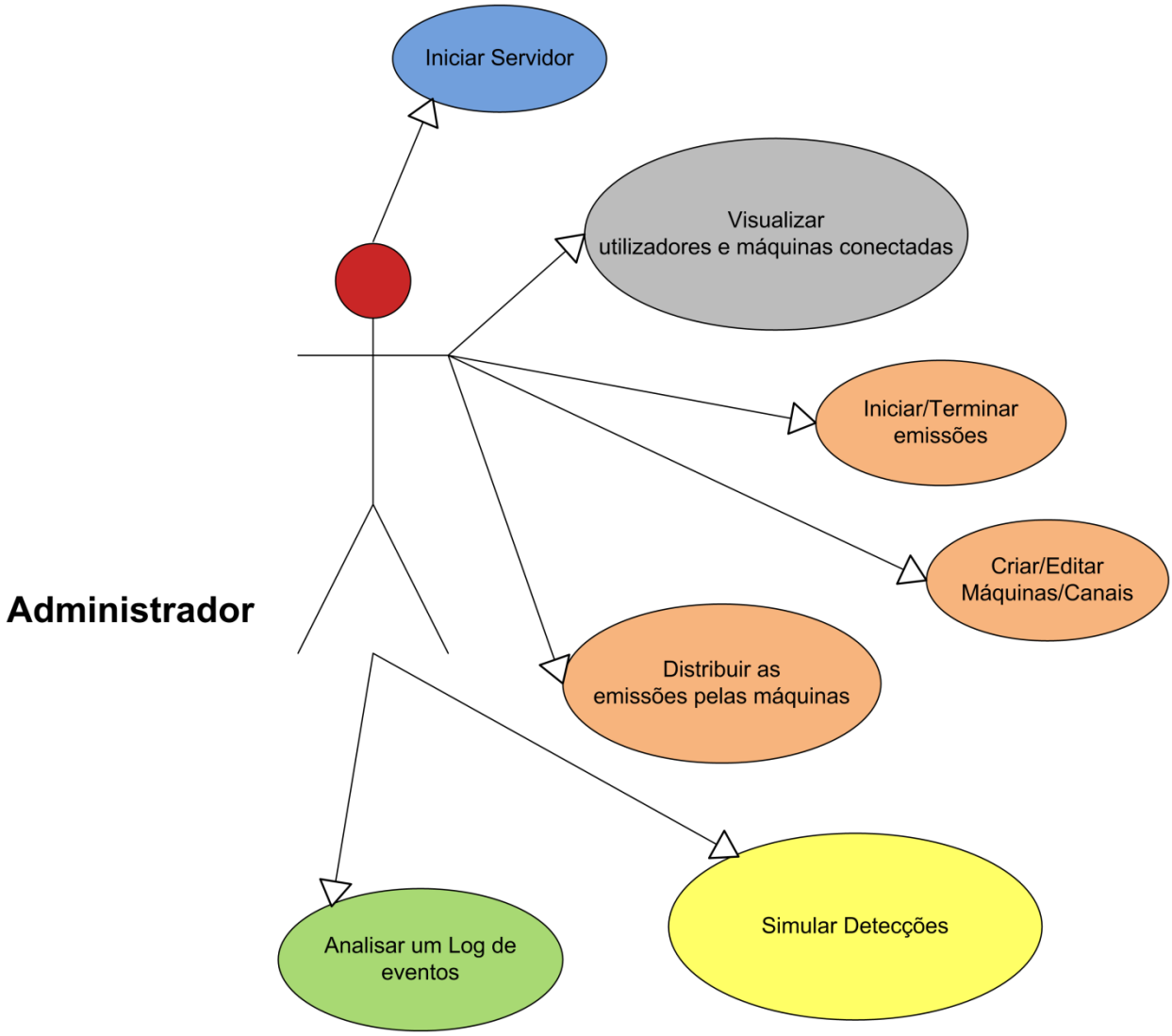

Figura 4.10 – Diagrama de Casos de Uso do *Back-End* do *LiveMeans*

Segue-se uma descrição dos diferentes casos de uso. Na *form* principal é possível:

- **Inicializar o servidor** – É estabelecida a conexão à base de dados e abertos os portos para a conexão das máquinas remotas (de *streaming* e de análise) e dos utilizadores.

- **Visualizar os utilizadores e máquinas conectadas** – O administrador pode aceder a uma lista dos utilizadores conectados através da aplicação *web* assim como das máquinas de análise conectadas ao *back-end*. Para além disso, permite visualizar uma lista de todas as máquinas de *streaming* existentes. Para tal, existe um jogo de cores utilizado:
  - Máquinas a vermelho – Máquinas que possuem uma referência na base de dados mas que não se encontram conectadas.
  - Máquinas a verde – Máquinas que possuem uma referência na base de dados e que se encontram, de facto, conectadas.
  - Máquinas a amarelo – Máquinas que se encontram conectadas mas que não possuem nenhuma referência na base de dados. Estas podem ser adicionadas à base de dados, mediante opção do administrador, assumindo a cor verde.
- **Visualizar canais e iniciar/terminar emissões** – O utilizador pode selecionar, na lista de canais, um conjunto de emissões, e inicia-las, caso não estejam a ser transmitidas, ou termina-las, caso estejam no ar. De notar que, para facilitar a selecção dos canais em causa, o administrador pode selecionar apenas algumas máquinas, de forma a que na lista de canais apenas sejam apresentados os canais pelos quais as máquinas selecionadas estão responsáveis. Mais uma vez, é utilizado o esquema de cores verde e vermelho para distinguir os canais que estão a ser transmitidos dos que não estão. Se um canal que está a ser transmitido for terminado de forma inesperada, a aplicação, ao receber a notificação da máquina de *streaming*, volta a apresentar o canal a vermelho, podendo o utilizador forçar, ou não, a sua retransmissão.
- **Simular detecções** – Para efeitos de *debug* e testes, o administrador pode simular as máquinas de análise, informando o sistema que uma determinada entidade foi reconhecida num dado canal.
- **Analisar o *log*** – O administrador pode analisar um registo que contém toda a actividade do *back-end*, registo de eventos e erros.

Já na *form* de administração de máquinas é permitido:

- **Criar máquinas e canais** – O administrador pode adicionar à base de dados, referências para novas máquinas de *streaming* ou configurar novos canais televisivos.
- **Edição/Remoção de máquinas/canais** – É possível modificar as propriedades (*ip* e porto no caso das máquinas, *url* e nome no caso dos canais) de máquinas e canais, ou até mesmo eliminá-los. No canto superior direito de cada máquina existe um botão que a elimina. Caso se pretendam eliminar canais, efectua-se o *drag and drop* dos mesmos para o *container* de canais desactivados e, já nesse container, selecionam-se os canais que se pretende eliminar e pressiona-se o botão existente no canto superior direito do *container*.
- **Distribuir as emissões** – Através de um sistema de *drag and drop* intuitivo e *user friendly*, o administrador pode definir quais os canais pelos quais cada máquina de *streaming* fica responsável.

Capítulo 5

Implementação – Aplicações de Apoio

Este capítulo pretende discutir as questões envolvidas na implementação das aplicações que, não fazendo directamente parte do sistema *LiveMeans*, possibilitam a sua existência, na medida em que permitem construir os modelos de som e imagem necessários para o seu funcionamento. Estas são descritas antes do próprio *LiveMeans*, uma vez que ajudam a compreender o fluxo de informação do sistema.

5.1 YouTube Flvs Crawler

Esta aplicação foi concebida na linguagem *Ruby*, uma vez que foi desenvolvida no início do projecto, altura em que se estava a aprender o funcionamento dessa linguagem. A aplicação começa por abrir uma página de pesquisa do *YouTube,* relativa às palavras-chave inseridas. A sintaxe do *url* das páginas de pesquisa do *YouTube* é a seguinte: "*http://www.youtube.com/results?search_query=Palavras-Chave*", sendo apenas necessário substituir "Palavras-Chave" pelas reais palavras-chave que se pretende.

De seguida, a aplicação utiliza um *parser* de *html*, designado por *hpricot*, para efectuar o *parsing* da página e obter os *urls* das páginas dos vídeos existentes nessa página de pesquisa. Após obter todas estas ligações, tipicamente vinte resultados por página, prossegue para a página seguinte, até se atingir o número de vídeos desejados.

Para obter, a partir do *url* da página de um vídeo, o *url* para o próprio *flv*, a aplicação utiliza um *script PHP* [TSL08] implementado com recurso a um servidor *Apache*. Após ter uma lista de *urls* dos vídeos que é necessário descarregar, são então efectuados todos os *downloads* para o repositório, criando-se uma pasta cujo nome resulta da concatenação das palavras-chave utilizadas para a pesquisa dos vídeos.

Finalmente, é criado na base de dados do sistema, um registo por cada vídeo descarregado, na tabela de vídeos não segmentados.

5.2 Segmentador de vídeos

Esta aplicação foi desenvolvida em C#, tendo sido utilizada a *frameWork .Net 3.5* embebida no *Visual Studio 2008*. É efectuada uma conexão entre a aplicação e a

base de dados do sistema, de forma a se aceder à tabela de vídeos não segmentados (eventualmente preenchida pelo *YouTube Flvs Crawler* ou por outra entidade) e à tabela que faz as associações entre estes vídeos e as entidades existentes na base de dados. De facto, cada vídeo pode estar associado a várias entidades, desde que os parâmetros *inPoint* e *outPoint,* que definem a região do vídeo que está a ser segmentada, sejam diferentes.

Sempre que um utilizador pede ao sistema um vídeo novo, o sistema retira, de forma aleatória, um vídeo da base de dados que ainda não tenha qualquer associação com alguma entidade, ou seja, que ainda não foi segmentado. Para ser retirado um vídeo da base de dados, de forma aleatória, basta acrescentar o seguinte à instrução *SQL*: *ORDER BY RAND() LIMIT 1*.

Umas das questões mais relevantes na implementação desta ferramenta prende-se com a gestão das regiões, ou seja, o processo de criação de uma nova região quando o utilizador prime o botão *cut*, ou a eliminação das mesmas.

- Se não houver nenhuma região existente, aquando da criação de um *keyPoint*, o sistema cria duas regiões, uma à esquerda do *keyPoint* e outra à direita.
- Caso já existam regiões, aquando da criação de um *keyPoint*, o sistema identifica a região em que o *keyPoint* está a ser criado e divide-a em duas. Para tal, são feitas duas tarefas:
    - Cria-se uma nova região, cujo *inPoint* é a posição do novo *keyPoint* e cujo *outPoint* é o mesmo do da região que se está a dividir.
    - Altera-se o *outPoint* da região já existente, para a posição do novo *keyPoint*.
- Quando se elimina uma região, o sistema elimina-a e altera o *inPoint* da região seguinte para a posição do *inPoint* da região eliminada, de forma que a região eliminada seja absorvida pela região seguinte.

De referir ainda que este tipo de segmentação que é feito de forma manual, no contexto do *Machine Learning*, é designado por *Ground Truth,* tendo assim uma grande credibilidade, sendo adequado à criação de modelos.

5.3 Reconhecimento facial

Sendo que os algoritmos da empresa apenas analisavam a componente sonora de um sinal audiovisual, investigou-se a possibilidade de integrar no sistema algoritmos de reconhecimento de faces. De facto, estes podiam ser utilizados, em conjunto com os algoritmos de reconhecimento de voz, para proporcionar uma maior eficácia e fiabilidade nos resultados através do cruzamento de informação.

O estudo das diferentes tecnologias disponíveis levou-nos a optar por uma desenvolvida pela *NeuroTechnology* [NT08], designada por *VeriLook.* Esta solução consiste num conjunto de bibliotecas e *dlls* com algoritmos de reconhecimento facial. Após o descarregar de uma versão de demonstração para 30 dias, foi necessário cerca de

uma semana para se conseguir adaptar todo o código, já que, apesar de os algoritmos estarem em *dlls* não acessíveis, o resto da aplicação possuía centenas de classes e milhares de linhas de código, pelo que a adaptação para inclusão no projecto foi bastante complexa.

Resumidamente, quando se criava um modelo, eram extraídos um *frame* por cada segundo de excerto de vídeo segmentado de uma determinada entidade. De seguida, estes algoritmos tentavam identificar uma face e, se conseguissem, as características desta eram adicionadas à base de dados, associadas à entidade em questão. Na fase de teste de um novo vídeo, voltava-se a extrair um frame por cada segundo de vídeo, extraíam-se as faces, e tentava-se fazer o *match* entre todas as faces detectadas no novo vídeo, e todas as faces encontradas na base de dados.

Após a adaptação, foram feitos vários testes ao projecto, no sentido de avaliar a sua eficácia. Em relação à *performance*, apesar do teste de um novo vídeo englobar um grande número de comparações (se fossem extraídas 30 faces do novo vídeo, houvessem 4 entidades, e cada uma delas tivesse 30 faces guardadas na base de dados, eram necessárias 3600 comparações), esta era bastante elevada, pelo que se concluiu que as comparações a fazer não envolviam grandes cálculos aritméticos.

No que toca à qualidade dos resultados, chegou-se à conclusão que, mesmo quando havia uma fraca qualidade de imagem, os algoritmos funcionavam bastante bem, desde que a face estivesse em posição frontal. Caso contrário, os resultados eram bastante piores. No sentido de ultrapassar esta lacuna, foram desenvolvidas algumas estatísticas para melhor aproveitar os resultados:

- Para cada entidade, somava-se as *confidences* (de 0 a 100) de cada *match* (entre cada uma das faces extraídas do novo vídeo e cada uma das faces da entidade guardadas na base de dados). De seguida, dividia-se este valor, pela soma de todas as *confidences* dos *matches* de todas as entidades, de forma a obter uma percentagem de probabilidade para cada entidade ser aquela presente no vídeo. Esta estatística já ajudava a melhorar os resultados do sistema mas continuava a existir um problema, relacionado com o facto de, quantas mais faces uma entidade tivesse na base de dados, mais probabilidade tinha de ter um maior número de *matches* no sistema.
- Tendo esse problema em mente, calculava-se também a probabilidade à priori, que cada entidade teria de ser aquela presente no vídeo, se o algoritmo de reconhecimento fosse aleatório. Para tal, bastava, para cada entidade, dividir o número de faces dessa entidade pelo número total de faces existentes na base de dados. Assim, se cada entidade tivesse um igual número de faces na base de dados, as probabilidades seriam semelhantes para todas as entidades. Utilizando estes dados, podemos agora analisar qual o desvio existente, da probabilidade à priori, para a probabilidade calculada. Entre as entidades que tivessem um desvio positivo, aquela cujo desvio representasse um maior aumento percentual, era a entidade retornada como mais provável.

A seguinte fórmula ilustra esta estatística:

$$\text{EntityDeviation} = \frac{(\Sigma\text{EntityMatchConfidence} / \Sigma\text{EveryMatchConfidence})}{(\text{EntityNumberOfFacesInDB} / \text{TotalNumberOfFacesInDB})} * 100 - 100$$

Com a utilização destas estatísticas, conseguiu-se melhorar os resultados obtidos, mesmo nas situações de inclinação da face. Contudo, os resultados continuavam a não ser os melhores. Apenas com quatro entidades, conseguia-se uma *accuracy* de cerca de 65%, o que, apesar de tudo, era melhor que os 25% que existiriam se o algoritmo fosse aleatório. De qualquer das formas, decidiu-se experimentar outras tecnologias, tendo-se utilizado uma grande variedade delas. Contudo, os resultados nunca ultrapassavam os obtidos com as bibliotecas *VeriLook.*

Decidiu-se então, investigar o porquê de os resultados não serem os mais satisfatórios, desenvolvendo-se de raiz algoritmos para reconhecimento de faces. Utilizando inicialmente o *Aforge,* uma *FrameWork* de *Computer Vision*, e depois o *OpenCV*, a *FrameWork* de *Computer Vision* da *Intel,* obtiveram-se alguns resultados interessantes. Começou-se por tentar detectar pontos na face, tal como o centro do nariz, as narinas, o centro da boca, etc. Conseguiram-se óptimos resultados utilizando algoritmos baseados em características *Haar*. Contudo, estes algoritmos *Haar* não apresentavam os melhores resultados quando a face se encontrava inclinada no plano frontal (sobre o eixo das ordenadas, num referencial de três dimensões). Para resolver este problema, e partindo do princípio que se identificava o centro dos olhos de uma face, calculava-se o declive da recta que ligava esses dois pontos. Posteriormente, calculava-se o ângulo de rotação que era necessário aplicar na imagem para essa recta tivesse declive zero e, consequentemente, eliminasse o factor de inclinação no plano frontal. Finalmente, bastava aplicar as transformações necessárias na imagem e voltar a calcular os pontos da face. Na Figura 5.1, é demonstrado o resultado de aplicação destes algoritmos, já após a rotação da imagem, sendo possível visualizar-se a recta de declive 0, que une os dois olhos, marcada a verde, assim como representações da localização do rosto, olhos, nariz e boca.

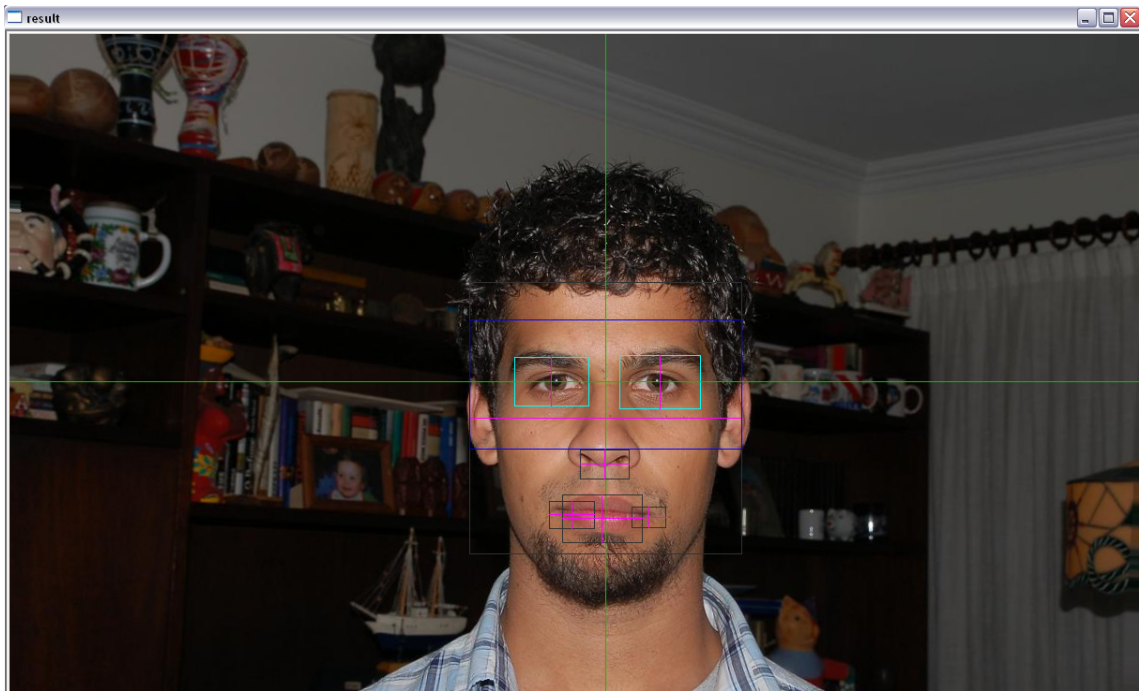

Figura 5.1 – Utilização do protótipo desenvolvido em OpenCv

Numa segunda fase, calcularam-se as distâncias entre todos os pontos, num esforço de tentar usar um grande número de distâncias de forma a anular os efeitos da inclinação em planos que não o frontal (há distâncias que não são alteradas com a inclinação sobre o eixo das cotas, tal como a distância vertical da linha do olhar até à linha das narinas, assim como distâncias que não são alteradas com a inclinação sobre o eixo das abcissas, tal como a distância horizontal entre os olhos). Após analisar os resultados de várias fotografias, a conclusão a que se chegou seria a esperada: Uma ligeira inclinação da face de uma pessoa, em planos que não o frontal (inclinações da face para os lados, para cima ou para baixo), faz com que o conjunto de distâncias entre os pontos da sua face se assemelhe mais a uma outra pessoa do que a ela própria em posição frontal.

Assim sendo, a conclusão foi óbvia: De facto, a única forma de obter resultados verdadeiramente interessantes seria a criação de modelos tridimensionais da face de cada entidade, fazendo-se o *matching* de novos vídeos ou fotos em partes desse modelo. Contudo, as aplicações existentes com base em tecnologia *3d*, fruto de vários anos de desenvolvimento, são extremamente dispendiosas, sendo vendidas a grandes clientes, como o estado Norte-Americano [HSW08]. Assim sendo, o melhoramento dos algoritmos foi deixado para trabalho futuro.

5.4 Model Creator

Esta aplicação, que permite a criação de modelos a partir da segmentação efectuada, é caracterizada por um conjunto de passos que culminam na criação dos modelos:

1. O utilizador insere as entidades para as quais deseja criar modelos.
2. O sistema retira, da base de dados, todas as referências de vídeos que tenham regiões segmentadas com alguma das entidades em questão.
3. Enquanto a aplicação vai percorrendo as várias referências da base de dados, encontra-se numa máquina de estados. Esta máquina de estados serve para criar uma estrutura que defina, para cada entidade, quais os vídeos que serão utilizados para a criação de modelos, e quais as regiões desses vídeos que vão ser realmente utilizadas. Este processo não é linear nem corresponde exactamente ao que está armazenado na base de dados, já que é necessário obedecer a algumas regras:
    a. Duração máxima de som que pode ser utilizada de um só vídeo – O objectivo é que um só vídeo não tenha um peso excessivo na construção de modelos sonoros, já que, por exemplo, no caso dos oradores, o som de fundo pode ser significativo e, como tal, devem-se usar vários vídeos de forma equilibrada, de forma a anular o ruído e destacar a voz do orador, já que assim esta se torna o único elemento comum entre todos os vídeos.
    b. Durações totais de amostras de som semelhantes para cada entidade – Pretende-se que a duração total de todas as amostras de som não varie muito de entidade para entidade, já que o contrário influencia os algoritmos no sentido de darem mais resultados positivos para as entidades que tinham uma maior quantidade de amostras aquando da criação dos modelos.
    c. Duração mínima de uma amostra de som – Este valor, tipicamente três segundos, pretende que não sejam utilizadas amostras excessivamente pequenas, já que estas não têm significado para os algoritmos.
4. Após se obter uma lista de todos os segmentos de vídeos que serão utilizados para criar os modelos de cada entidade, utiliza-se a ferramenta *ffmpeg* para dois fins:
    a. Extrair o som de todos os segmentos de vídeos e convertê-lo para ficheiros *wav* com um *bit rate* de 22050.
    b. Extrair um *frame* por cada segundo dos segmentos de vídeos.
5. Os algoritmos da empresa são utilizados para criar modelos, a partir de todos os excertos de som criados para cada entidade.
6. Caso a entidade que se está a tratar seja um orador, as bibliotecas modificadas da *VeriLook* são utilizadas para tentar extrair faces de cada *frame* obtido a partir dos segmentos de vídeos, armazenando-se as características de todas as faces na base de dados.

5.5 Model Tester

À semelhança do *Model Creator,* o processo que leva a que se obtenham *matches* a partir de um novo vídeo, é dividido em vários passos:

1. Utilizar o *ffmpeg* com os mesmos objectivos pelos quais o *Model Creator* utilizava essa aplicação:
    a. Extrair o som do vídeo e convertê-lo para um ficheiro *wav* com um *bit rate* de 22050.
    b. Extrair um *frame* por cada segundo do vídeo.
2. Efectuar o *match* dos conteúdos do vídeo com alguma das entidades para as quais foram construídos modelos:
    a. Os algoritmos da empresa são utilizados para tentar, com os modelos já criados, reconhecer a entidade presente, a partir no som extraído do novo vídeo.
    b. As bibliotecas alteradas da VeriLook e um conjunto de heurísticas são utilizadas para reconhecer a entidade presente, caso se trate de um orador, a partir da extracção de faces do novo vídeo e tentativa de *match* com as faces existentes na base de dados. Este processo é explicado em detalhe no capítulo 5.3.
3. Os resultados provenientes da análise de vídeo são cruzados com os provenientes da análise de som e são devolvidos ao utilizador.

Capítulo 6

Implementação – LiveMeans

Apesar de as outras aplicações descritas serem fundamentais para o funcionamento do sistema *LiveMeans*, na medida em que possibilitam a criação de modelos que são utilizados pelos algoritmos da empresa para reconhecimento de entidades nos conteúdos audiovisuais, estas não são parte integrante do sistema. De facto, efectuam o trabalho de *background* que é necessário ser feito para a aplicação poder ser utilizada, sendo que, após esse trabalho, deixam de ser utilizadas até ser necessário um aumento ou evolução dos modelos ou da base de dados de audiovisuais existente. Assim sendo, importa distinguir estas aplicações do restante sistema designado por *LiveMeans*, em que todos os módulos são constantemente utilizados no fluxo normal de uso da aplicação.

É ainda importante salientar que o sistema *LiveMeans* é um sistema complexo e que vários módulos do mesmo não foram desenvolvidos no decorrer deste projecto. Esses módulos são os seguintes:

- Algorítmos de análise de som, propriedade da *ClusterMedia Labs*.
- Modificação do servidor de *RTMP Red5* para possibilitar o *MultiStream.*
- *Front-end* gráfico, desenvolvido por uma equipa de designers sub-contratada.
- *Timeline SIMILE,* para visualização dos eventos de forma cronológica. Módulo desenvolvido por um departamento do *M.I.T.*

Já as contribuições para o sistema, podem ser analisadas no capítulo 1.5.

6.1 Arquitectura Geral

O diagrama de arquitectura, presente na Figura 6.1, demonstra todos os componentes envolvidos neste sistema. Este diagrama é baseado no diagrama de arquitectura proposto no capítulo 4 mas, desta vez, encontra-se completo e mapeado às tecnologias.

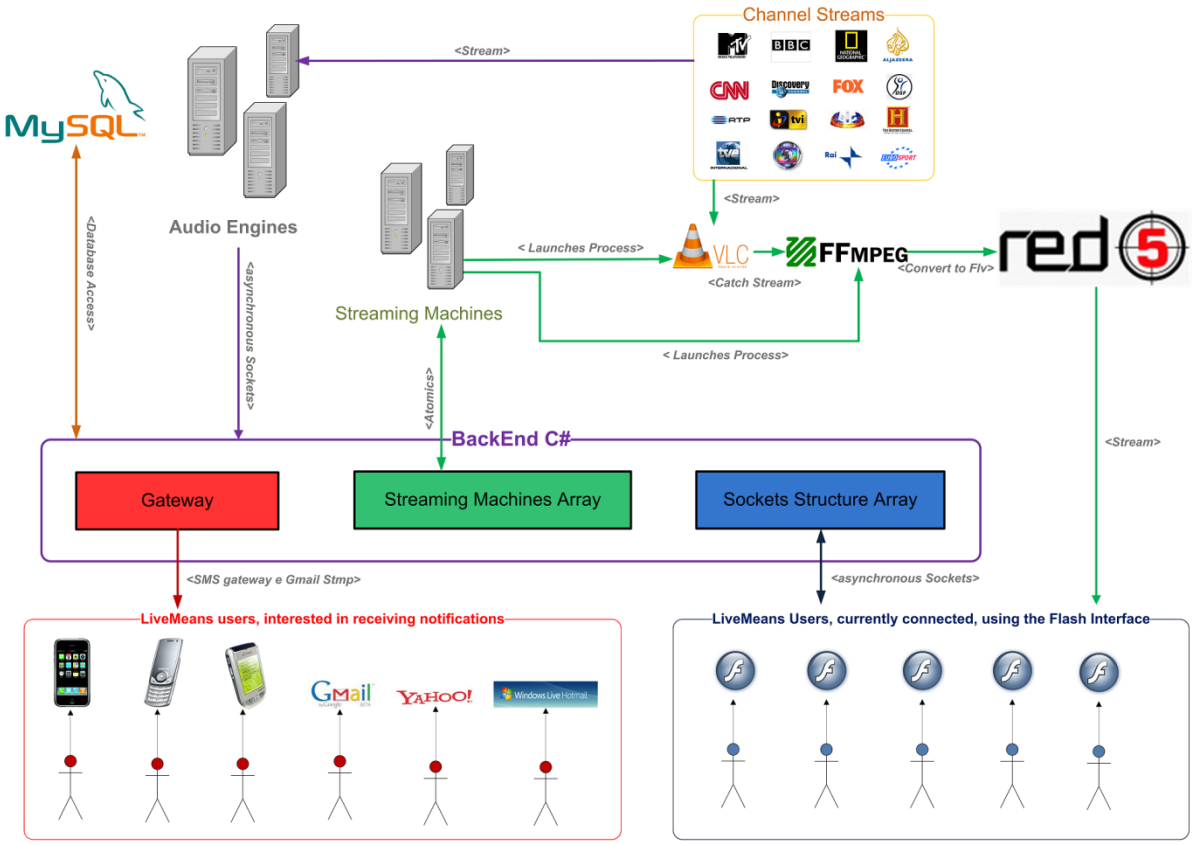

Figura 6.1 – Diagrama de Arquitectura do *LiveMeans*, completo e mapeado a tecnologias

O *back-end* do sistema, sendo a sua peça central e fundamental, é constituído por três módulos principais:

1. **Gateway** – Este módulo (a vermelho no diagrama) é responsável pela notificação remota de utilizadores. Sempre que um conteúdo é detectado, quer numa emissão *Live*, quer num ficheiro enviado por um utilizador, este módulo é invocado e é responsável por obter, da base de dados, todos os utilizadores interessados no conteúdo detectado. De seguida, notifica-os remotamente, de acordo com as suas preferências, por *sms* ou *e-mail*.
2. **Client Sockets –** Este módulo (a azul no diagrama) é responsável por toda a comunicação com os clientes conectados através do *front-end* gráfico desenvolvido em *Flash*. Toda a comunicação entre o *back-end* e o *front-end* é efectuada neste módulo, a partir de um protocolo de mensagens definido, tais como o *login,* envio de alertas, notificações, pedidos do cliente, etc.
3. **Streaming –** Este módulo (a verde no diagrama) é responsável pela gestão de todas as emissões *live* para os clientes *Flash,* e pela gravação dos conteúdos. É este módulo que distribui as emissões por todas as máquinas remotas disponíveis, trocando mensagens com elas no sentido de iniciar ou terminar a emissão de determinados canais ou de iniciar a gravação de um determinado conteúdo para futura visualização (*Video On Demand*).

Para além destes três módulos principais, presentes de forma explícita no diagrama, existem outros fundamentais para o funcionamento da aplicação:

4. **Audio Engines –** Módulo responsável pela gestão e comunicação com os *Audio Engines* que correm os algoritmos da empresa para reconhecimento de entidades nas emissões *Live*. Na prática, é este módulo que interpreta os resultados devolvidos pelos *Audio Engines*, invocando de seguida o módulo *Client Sockets* para notificações no *front-end* gráfico, e o módulo *GateWay* para notificações remotas.
5. **Machines Administration** – Módulo que providencia uma *interface* gráfica para gestão, em tempo real, das máquinas remotas. Permite o registo de novas máquinas, a edição e eliminação das mesmas ou a criação, edição e remoção de canais *Live*. Permite ainda, com um sistema intuitivo de *drag and drop*, definir os canais que cada máquina fica responsável por transmitir.
6. **Módulo Video On Demand** – Este módulo consiste numa integração da aplicação *Model Tester*, já descrita no capítulo 5.5, no *back-end* do sistema. É responsável então pelo *upload* de um ficheiro por parte do utilizador, e consequente análise com os algoritmos da empresa e as bibliotecas alteradas da *VeriLook* para reconhecimento de faces. Após a análise, os resultados são enviados para o utilizador e o ficheiro é transferido para o servidor de *Video On Demand*, notificando-se todos os utilizadores interessados nos conteúdos detectados, através dos módulos *Client Sockets* (notificações no cliente *Flash)* e *Gateway* (notificações remotas).
7. **Cryptography –** É responsável por todo o sistema de criptografia do *LiveMeans*, em especial pela encriptação e desencriptação de dados com o algoritmo *RSA* e pela encriptação de dados com o algoritmo *MD5*.
8. **Communication Protocol –** Define todo o protocolo, formato e tipos de mensagens trocadas entre o *back-end* e as restantes aplicações que fazem parte do *LiveMeans*:
    a. Clientes *Flash*
    b. *AudioEngines*
    c. *Streaming Machines*
9. **Debug Functions** – Funções de *Debug* e administração, tal como o *reset* dos clientes conectados, a visualização do estado das máquinas remotas, estado da ligação à base de dados, etc.
10. **Auxiliary Functions –** Funções auxiliares utilizadas pelo sistema.

Nas próximas secções, alguns destes módulos irão ser detalhados de forma mais pormenorizada, sendo explicadas algumas opções e decisões de implementações tomadas.

6.2 Base de dados

A base de dados da aplicação, composta por trinta tabelas, é comum não só aos módulos do *LiveMeans* mas também às aplicações adicionais, *YouTube Flv Crawler,* Segmentador de vídeos, *Model Creator* e *Model Tester.* O diagrama de base de dados, presente na Figura 6.2, apresenta todas as tabelas a respectivos campos, chaves e relações.

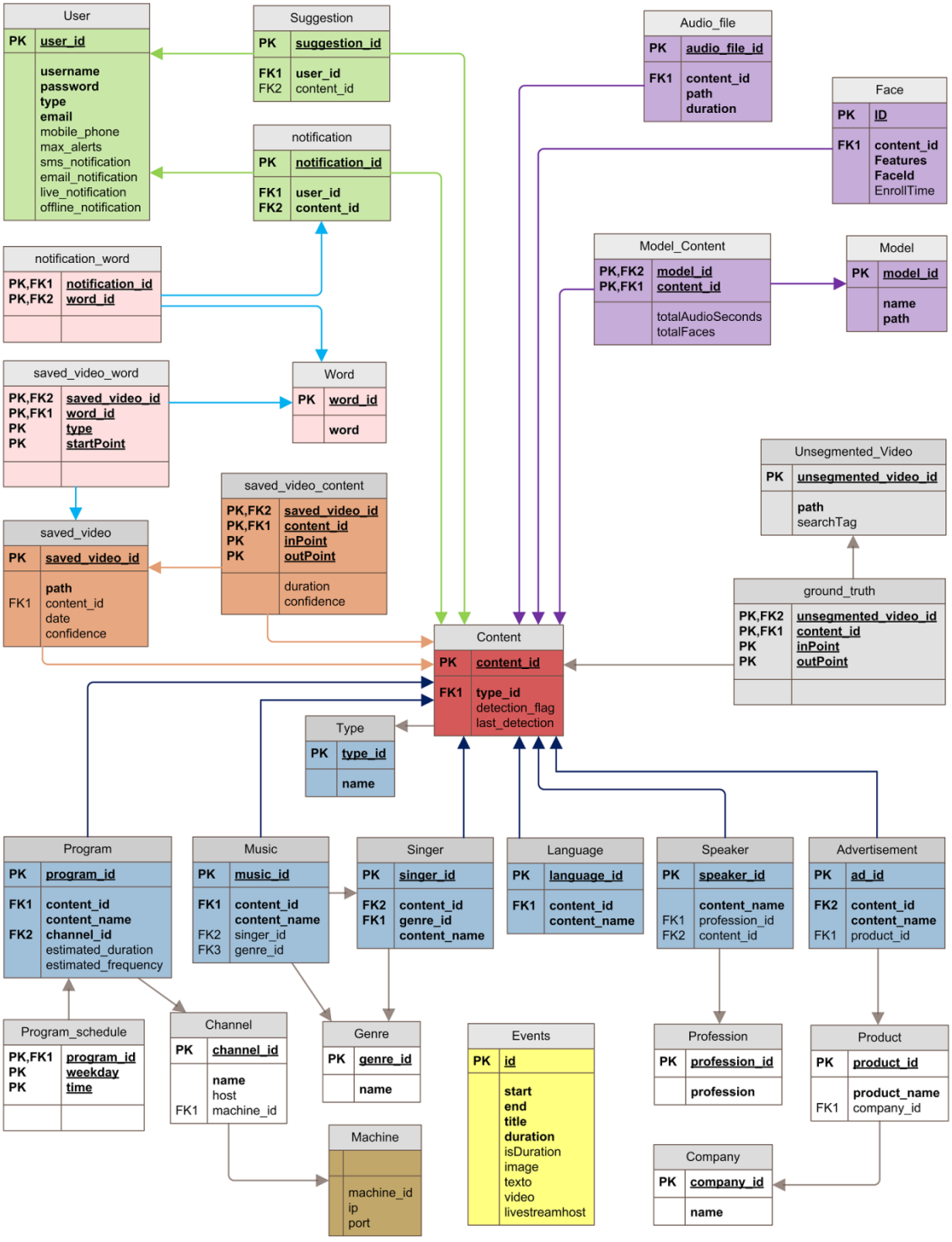

Figura 6.2 – Diagrama de Base de Dados do *LiveMeans*

Dada a complexidade da base de dados, e de forma a auxiliar a sua interpretação, foram constituídos grupos, visíveis no diagrama pelo conjunto de cores utilizado. Segue-se de seguida, uma descrição geral de cada conjunto, tentando-se reproduzir, tanto quanto possível, a ordem pela qual os conjuntos são utilizados num fluxo normal de utilização:

- **Conjunto Vermelho** – Constituído pela tabela central da base de dados, *Content*, que representa uma qualquer entidade, que pode ser de vários tipos (tais como orador ou música). A importância desta tabela pode ser defendida pelo facto de ser referenciada com *foreign keys* por outras quinze tabelas.
- **Conjunto Azul** – Define os tipos possíveis de entidades. Estes podem ser programas, músicas, cantores, linguagens, oradores ou publicidade. O facto de existir uma tabela por cada tipo de entidade prende-se com o facto de cada tipo possuir campos diferentes. Por exemplo, uma entidade do tipo *Advertisement* pode estar associada a um produto, enquanto que uma entidade do tipo *Speaker* pode estar associada a uma profissão. De notar ainda a existência de uma outra tabela do conjunto azul, a *Type*, referenciada também pela tabela *Content*, que define qual o tipo de conteúdo que caracteriza essa entidade. Assim, se o sistema quiser analisar um determinado conteúdo, verifica qual é o seu tipo a partir da referência na tabela type e, de seguida, obtém da tabela do conjunto azul que representa esse tipo, o registo para o conteúdo, podendo ainda analisar os campos das tabelas do conjunto branco, associados à entidade, que também a caracterizam.
- **Conjunto Branco** – Definem as propriedades específicas de cada tipo de entidade:
    a. Um programa pode ter associado um horário previsto para ser emitido assim como um canal de televisão.
    b. Uma música pode estar associada a um género musical ou a um cantor.
    c. Um cantor pode estar associado a um género musical.
    d. Um orador pode estar associado a uma profissão.
    e. Um anúncio pode estar associado a um produto, que por sua vez pode estar associado a uma empresa.
- **Conjunto Cinzento** – Estas tabelas são utilizadas unicamente por aplicações anexas ao *LiveMeans*:
    a. O *YouTube Flv Crawler*, ou outra aplicação de descarregamento de vídeos, preenche a tabela *Unsegmented_Video*, com todos os vídeos não segmentados que foram descarregados.
    b. Um utilizador pode então utilizar a aplicação Segmentador de Vídeos, que preenche então a tabela *ground_truth*, que faz as associações entre os vídeos não segmentados e as entidades. De notar que esta tabela resolve uma situação *muitos-para-muitos* já que um determinado vídeo pode conter associações a várias entidades, desde que sejam definidas as regiões do vídeo em que cada uma surge.
- **Conjunto Roxo** – Estas tabelas são utilizadas também por aplicações anexas ao *LiveMeans,* sendo que este apenas a usa no módulo *Video On Demand:*
    a. Quando se cria um modelo, com a aplicação *Model Creator*, é criada uma referência na tabela *Model*, assim como as associações a todas as entidades a que esse modelo diz respeito, utilizando a tabela *Model_Content*.
    b. O *Model Creator* cria ainda referências nas tabelas *audio_file* e *face* para todos os excertos de som e faces extraídas aquando da criação do modelo, associando cada um desses registos à entidade que caracterizam.

c. Estes dados são utilizados pelo módulo *Video On Demand* do *LiveMeans* para efectuar a análise semântica aos vídeos novos enviados pelos utilizadores.

- **Conjunto Verde** – Este conjunto é responsável pela gestão dos utilizadores:
  a. A tabela *User* armazena toda a informação de um utilizador, em especial, as suas preferências em relação a notificações.
  b. A tabela *Notification*, cria associações entre utilizadores e entidades, de forma a definir quais as entidades em que cada utilizador está interessado e deseja receber notificações da sua detecção.
  c. A tabela *Suggestions* permite que um utilizador sugira a criação de modelos para reconhecimento de determinadas entidades ainda não existentes.
- **Conjunto Salmão** – Este conjunto é responsável pelo armazenamento de informação relativa aos vídeos enviados pelos utilizadores. É composto pelas tabelas *saved_video* e *saved_video_content*, funcionando de uma forma bastante semelhante às tabelas *unsegmented_video* e *ground_truth* do conjunto cinzento, sendo que as únicas diferenças prendem-se com a origem dos vídeos e com o facto de a segmentação ser feita de forma manual no conjunto cinzento, e de forma automática no conjunto salmão.
- **Conjunto cor-de-rosa** – Este conjunto é responsável pelo registo de palavras-chave que são ditas em cada vídeo gravado. Os algoritmos da empresa ainda não fazem esta análise de palavras, mas fica em aberto uma incorporação de algoritmos, eventualmente utilizando a plataforma *SAPI* da *Microsoft,* para segmentação de vídeos por palavras.
- **Conjunto Amarelo** – Constituído apenas pela tabela *Events*, regista todas os reconhecimentos efectuados, de conteúdos *live* ou não, registando ainda a localização dos vídeos gravados e a hora de detecção. Estes dados podem ser utilizados por uma *timeline* existente na empresa, desenvolvida por um departamento do *M.I.T.,* que permite visualizar numa sequência cronológica todos os vídeos gravados.
- **Conjunto Castanho** – Constituído apenas pela tabela *Machine*, define as propriedades de cada máquina remota utilizada para *streaming* das emissões. Em especial, define o seu *ip* e o porto pelo qual se pode aceder ao servidor de *RTMP* do *Red5* para visualização dos canais pelos quais esta máquina está responsável. Este porto não é necessariamente o porto real da máquina, já que se esta estiver numa rede, passa-se a registar o porto público do *router*, responsável pelo reencaminhamento para o porto *RTMP* dessa máquina. De notar que a tabela *channel*, apesar de pertencer ao conjunto branco, já que é uma propriedade de um programa de televisão, também está relacionada com este conjunto castanho, na medida em que cada canal pode ter uma *foreign key* a referenciar uma máquina, criando-se associações e definindo-se os canais pelos quais cada máquina de *broadcast* fica responsável.

6.3 Segurança e Autenticação

Uma das questões essenciais, e à qual foi dada bastante relevância, foi a questão da segurança e autenticação. Acima de tudo, existiam as seguintes necessidades:

1. Guardar as *passwords* dos utilizadores de forma encriptada na base de dados.
2. Nunca enviar a *password* de forma desprotegida pela rede.
3. Garantir que o utilizador sabe, de facto, a *password*, e não apenas a encriptação desta.

A primeira aproximação efectuada para resolver estes problemas foi guardar a *password* na base de dados, encriptada com o algoritmo *MD5*. Assim sendo, aquando do *Login,* o utilizador enviava a *password*, o *back-end* encriptava-a com o algoritmo *MD5*, e comparava-a com a *hash* da base de dados. Contudo, este sistema não respeitava as necessidades 2 e 3. Assim sendo, o cliente passou a enviar a *password* já encriptada, comparando-a então o *back-end* directamente com a *hash* guardada na base de dados. Ainda assim, esta solução não garantia a necessidade 3:

a) Um utilizador podia, com a utilização de um *sniffer*, captar os dados enviados entre o cliente e o servidor, ou seja, a *hash MD5* da *password.*
b) Conseguindo aceder ao código da aplicação *Flash,* poderia então modifica-la para enviar directamente essa *hash*.
c) Assim, poderia efectuar o *login,* com a conta de outro utilizador, sem sequer saber a sua *password*.

Para resolver a preocupação 3, optou-se por uma solução final:

a) As *passwords* passaram a ser gravadas com um algoritmo que não perde informação, e que permite a desencriptação das mesmas mediante a utilização de chaves secretas, apenas existentes no servidor. Para tal foi escolhido o algoritmo *RSA*.
b) O *Login* passou a efectuar-se em duas fases:
    a. A primeira fase é semelhante à solução anterior. Ou seja, o utilizador insere a sua *password*, o cliente *flash* encripta-a com o algoritmo *MD5* e envia-a através da rede. De seguida, o *back-end*, utilizando as suas chaves *RSA*, desencripta a *password* do utilizador na base de dados, encripta-a novamente, desta vez com o algoritmo *MD5,* e compara as duas *hashes*. Se forem iguais, o processo de *login* avança para a segunda fase.
    b. A segunda fase permite garantir a preocupação 3, ou seja, garantir que o utilizador sabe, de facto, a sua *password*. Para tal, após o término da primeira fase, o *back-end* envia para o cliente um conjunto de caracteres gerado aleatoriamente, designado por *salt.* De seguida, a aplicação *Flash* concatena este *salt* com a *password* inserida, encripta-a com o algoritmo *MD5* e envia essa *hash* para o servidor. O servidor, para validar esta *hash*, retira da base de dados a *hash RSA* da *password* do cliente,

desencripta-a com a chave *RSA,* concatena-lhe o *salt*, encripta-a com o algoritmo *MD5* e compara então as duas *hashes*. Se forem iguais, a autenticação é concluída com sucesso.

Com esta solução, garantimos que as três necessidades de segurança são conseguidas. Mesmo que um utilizador utilize um *sniffer* para captar os pacotes trocados entre o cliente servidor, e consiga ainda aceder ao código do cliente *Flash*, não poderá fazer nada com essa informação, já que necessita de saber a *password* do utilizador para lhe poder concatenar o *salt* e encriptar novamente. Saber a *Hash MD5* não é suficiente para efectuar o *login.*

Para tornar o sistema de *login* ainda mais robusto, desde que é efectuada a conexão, que só é feita, por parte do cliente *flash,* quando o utilizador insere o seu nome de utilizador e palavra-chave, são atribuídos um máximo de dois minutos para que todo o processo de *login* esteja completo, antes de fechar a conexão com o cliente. Para além disso, após o envio do *salt,* do servidor para o cliente, este dispõe apenas de dez segundos para devolver a *hash MD5* adequada.

Poderia ser argumentado que esta solução não necessita da primeira fase, uma vez que a segunda é suficiente para garantir as três necessidades de segurança. De facto, o servidor poderia enviar uma mensagem para o cliente, aquando do estabelecimento da conexão, com o código *salt*, e o cliente responder já com a *hash MD5* final. Contudo, existem motivos para se ter preferido a validação em duas fases. Estes motivos relacionam-se com o facto de ser importante distinguir uma falha de autenticação por motivos de esquecimento da *password* ou por motivos ilícitos. Mais especificamente, o que importa salientar é que o sucesso da primeira fase de autenticação depende do utilizador enquanto que o sucesso da segunda fase depende apenas da aplicação *flash*. Assim, caso a autenticação falhe na primeira fase, é fundamental dar ao utilizador mais oportunidades de inserir os dados de autenticação. Já no caso de a autenticação falhar na segunda fase, torna-se óbvio que alguém está a tentar entrar no sistema de forma ilícita, muito provavelmente utilizando um cliente alterado. Nessa situação, é interessante guardar o registo da tentativa de *hacking* e, eventualmente, criar uma *blacklist* de *ips* utilizados para o efeito.

6.4 Módulo Vídeo On Demand

A implementação deste módulo foi bastante simples já que todo o trabalho tinha sido já feito no desenvolvimento do *Model Tester*, cuja implementação é discutida no capítulo 5.5. Após a adaptação do *Model Tester* para se tornar um módulo do *LiveMeans*, foram necessários os seguintes ajustes adicionais:

- Criação de uma página *server-side*, desenvolvida em *ASP,* que permite o *upload* de um ficheiro do cliente *flash* para o servidor. Para tal, foi necessário configurar um servidor *IIS*. O cliente, para além de fazer *upload* do ficheiro, envia um código aleatório para identificação. O servidor concatena esse código aleatório ao nome do ficheiro.

- Após o *Upload* do ficheiro, o cliente envia uma mensagem para o servidor com o nome do ficheiro e o código gerado. O servidor verifica se o código é válido e se for, inicializa a análise:
    - À semelhança do que é feito no *ModelCreator*, após a extracção do som e *frames*, utilizando o *ffmpeg*, os algoritmos da empresa são utilizados para a análise do som enquanto que as bibliotecas alteradas da *VeriLook* são utilizadas para fazer o *match* das faces.
    - Os resultados são então cruzados e o vídeo é segmentado na base de dados de acordo com esses resultados.
    - Uma cópia do vídeo é copiada para o servidor de *Video On Demand*.
    - Os módulo *Gateway* e *Client Sockets* são invocados de forma a notificar os utilizadores interessados nas entidades detectadas. O primeiro através do *front-end* gráfico e o segundo de forma remota por *sms* ou *e-mail.* Os utilizadores podem então visualizar o vídeo no cliente gráfico na secção de *Video On Demand.*

6.5 Gateway

Este módulo, como já foi descrito, notifica clientes remotamente acerca de determinados conteúdos em que estes estão interessados e que foram detectados. De acordo com as preferências de cada utilizador, existem duas formas para efectuar estas notificações:

- E-mail – Foi criada uma conta de *e-mail* nos servidores de *mail* do *google*, clustermedia@gmail.com, utilizada para enviar este tipo de notificações. Sempre que é necessário enviar um *e-mail* de notificação, acede-se ao servidor de *SMTP* do *google*, efectua-se o *login* com os dados da conta, e envia-se o *e-mail* para o destinatário.
- *SMS* – Foi integrada no projecto uma *Gateway* de *sms* disponibilizada pela empresa *Shortcut* [SHO08]. Esta empresa, sediada em Matosinhos, disponibiliza vários tipos de serviços a empresas, entre eles o envio de *sms*. Foi fornecido, por parte da empresa, um *snippet* de código em *C#*, que em conjunto com os dados de acesso ao servidor, acede à *gateway* de *sms*. Esta conexão é efectuada no arranque do servidor. Sempre que é necessário enviar uma *sms,* é feito um pedido à *gateway* que trata então de entregar a mensagem aos destinatários.

6.6 Sockets assíncronas

Toda a comunicação efectuada entre o *back-end* e os restantes módulos é feita através da utilização de *sockets* assíncronas. Os primeiros desenvolvimentos do *back-end* acentavam na utilização de *web services,* contudo, existiam situações para as quais os *web services* tinham uma eficácia muito baixa. Estas situações eram aquelas em que

era necessário o envio de uma mensagem do servidor para o cliente, sem que este a tivesse requisitado, já que para tal era necessário efectuar um *polling* constante ao servidor no sentido de saber se houve modificações. No capítulo 3.2 são discutidas as vantagens e desvantagens de cada tecnologia, e os motivos que levaram à adopção das *sockets* assíncronas.

No que toca à implementação das *sockets* assíncronas, o processo é semelhante em todos os casos, quer se trate da comunicação com o *front-end* gráfico, desenvolvido em *flash,* com os *AudioEngines*, desenvolvidos em *C++*, ou com o *Atomic Remote Process Manager,* desenvolvido em *C#*:

- O servidor cria uma *socket,* que escuta conexões num porto conhecido pelo cliente. De seguida define uma função de *callback*, invocada quando é recebida uma conexão.
- Quando uma conexão é recebida, a função de *callback* é despoletada. É então criada uma estrutura para o cliente que se conectou. Esta estrutura varia consoante o tipo de cliente que se ligou, ou seja, se é um cliente *Flash*, um *AudioEngine* ou uma *Streaming Machine*, e é adicionada, consoante o caso, à *ArrayList FlashClients, AudioEngines* ou *StreamingMachines*. Estas *ArrayLists* são então responsáveis por conter referências para todas as estruturas de conexão existentes, de forma a facilitar a iteracção de todas as estruturas de um dado tipo.
- Cada estrutura de conexão contém várias variáveis, *flags* e funções, sendo algumas delas, comuns aos três tipos de estruturas:
    - Uma *socket* para comunicação entre a máquina conectada e o *back-end.*
    - Um *buffer* de escrita e um *buffer* de leitura do que é enviado pela *socket.*
    - Uma função de *callback*, despoletada sempre que são recebidos dados na *socket.* Esta função interpreta as mensagens recebidas, de acordo com o protocolo estabelecido, e efectua as acções necessárias.
- Quando uma conexão é perdida, a estrutura é eliminada e removida da *ArrayList* que a refere.

6.7 Sistema Distribuído para Gestão de Streamings

Desde cedo se percebeu que a emissão de um *LiveStream* requer bastante processamento. De facto, como é discutido no capítulo 3.6, para garantir a emissão dos *LiveStreams*, é necessário um servidor de *RTMP*, e, por cada emissão, uma instância das aplicações *Ffmpeg* e *VLC.* Assim sendo, e após a realização de alguns testes de *benchmarking,* chegou-se à conclusão que, numa das máquinas para *broadcast* existentes na empresa, um número de emissões superior a cinco criava sérios problemas de *performance*.

6.7.1 Atomic Remote Process Manager

Para resolver este problema, foi necessário distribuir o *broadcast* das emissões por várias máquinas. Para se conseguir este sistema distribuído, foi desenvolvida uma aplicação, designada por *Atomic Remote Process Manager*, que se instalava em cada máquina de *broadcast.* Esta aplicação, instalada numa máquina com um servidor *RTMP* a correr, conecta-se então ao servidor, via *sockets*, estando então disponível para iniciar ou terminar instâncias do *vlc* e do *ffmpeg*. É então o servidor que, tendo em conta as máquinas de *broadcast* conectadas, distribui a emissão de canais pelas mesmas, enviando uma mensagem com os parâmetros com que devem ser invocados o *vlc* e o *ffmpeg* de forma a captar correctamente a emissão e enviá-la para o servidor de *RTMP*.

Para além destes parâmetros, o servidor envia ainda um *id* único, que caracteriza a emissão e que é alojado na estrutura do canal, quer do lado do servidor quer do lado do *atomic remote process manager,* possibilitando um nível mais avançado de interacção:

- O servidor pode enviar uma mensagem para uma máquina de *broadcast* no sentido de terminar a emissão de um canal, bastando para isso referenciar o *id do canal.* O *atomic remote process manager* apenas necessita de terminar as instâncias do *vlc* e *ffmpeg* responsáveis por esse canal.
- O *atomic remote process manager*, sempre que inicia a emissão de um canal, monitoriza o decorrer dos processos lançados. Caso estes sejam terminados por algum motivo, por exemplo, por o *url* da emissão não existir ou por esta ter sido interrompida, o servidor é notificado do facto da emissão desse canal ter terminado de forma inesperada, fazendo então os ajustes necessários na estrutura do canal ou forçando a retransmissão.

6.7.2 Interfaces Gráficas do Back-End

De forma a melhor aproveitar o potencial resultante do desenvolvimento de um sistema distribuído, foram feitas algumas modificações gráficas no *back-end*, descritas no capítulo 4.3.2. É a comunicação entre o *back-end* e os *atomic remote process managers* que permite então ao administrador interagir com as máquinas, iniciar ou terminar as emissões e, visualizar, num esquema de cores, quais as máquinas que se encontram conectadas e canais que se encontram activos.

Foi ainda criada uma nova *form* para administração, que permite a criação, edição e remoção de máquinas e canais assim como a distribuição dos canais pelas diferentes máquinas. Caso não se deseje eliminar um canal, apesar de não se querer que este seja transmitido por nenhuma máquina, pode-se coloca-lo num *container* para canais desactivados.

Uma das preocupações existentes aquando do desenvolvimento desta *form* de administração de máquinas foi a usabilidade:

1. Sistema de *drag and drop* – Permite que todos os canais sejam "arrastados" de um container para outro. Para tal, foi necessário a utilização dos controlos *ListView* em vez de *ListBox*, já que estes possuem mais opções e eventos que facilitam a implementação do *drag and drop*. Ainda assim, alguns pormenores de implementação revelaram-se morosos.
2. Dinamismo dos *containers* – Aquando da inicialização da *form*, são criados tantos *containers* quantas as máquinas existentes na base de dados. É ainda criado um *container* extra para os canais desactivados. O dinamismo referido prende-se com o facto de, através da criação de estruturas próprias, se permitir a criação e eliminação de máquinas, sendo que:
    a. Quando uma máquina é eliminada, todas as outras são movidas para trás, de forma a ocupar o vazio provocado pela eliminação.
    b. Quando uma máquina é inserida, é desenhada a seguir à ultima máquina existente, ou, caso não haja espaço, é desenhada numa posição inferior, alinhada à esquerda.

Um dos grandes desafios desta *form* de administração de máquinas e canais foi garantir que, quando eram feitas alterações no esquema de distribuição, o sistema se mantinha em execução normal, sendo apenas terminadas as emissões que fossem estritamente necessárias, ou seja, aquelas cujos canais foram eliminados ou transferidos para uma máquina diferente, no novo esquema de distribuição. Assim, conseguiram-se os seguintes objectivos:

- Se um canal que se encontra a ser transmitido, for eliminado ou transferido para uma outra máquina de *broadcast* no novo esquema de distribuição, é primeiro enviada uma mensagem para a máquina que o está a transmitir, no sentido de terminar a emissão.
- Se uma máquina que se encontra conectada, for eliminada no novo esquema de distribuição, é necessário terminar todas as emissões dos canais que ela está a transmitir, e, de seguida, fechar a conexão com a máquina.
- Se uma máquina está conectada, e continua a existir no novo esquema de distribuição, então é necessário manter a conexão com essa máquina.
- Se um canal, que está a ser transmitido, continuar presente na mesma máquina, no novo esquema de distribuição, é necessário manter a emissão do canal.

Para se conseguirem estes objectivos, e tendo em conta que as estruturas gerais de canais e máquinas do sistema *LiveMeans* são diferentes das estruturas temporárias existentes na *form* de administração, a função de gravar as modificações no esquema de distribuição é constituída pelos seguintes passos:

1. É feita uma validação dos *ips* e portos inseridos para cada máquina. Se existir algum *input* incorrecto, é pedido ao utilizador que o corrija.

2. Obtém-se uma lista de todos os canais que estão, de momento, a ser emitidos, e das máquinas que se encontram conectadas.
3. Para cada canal que está a ser emitido, guarda-se numa estrutura temporária, designada por *RunningChannelsThatWillContinueRunning,* aqueles que devem continuar a ser emitidos após as alterações, ou seja, aqueles que no novo esquema de distribuição não foram eliminados e mantêm-se na mesma máquina de *broadcast.* Os restantes são armazenados numa outra estrutura temporária, designada por *RunningChannelsThatWillBeClosed.*
4. Para cada máquina que está conectada, guarda-se numa estrutura temporária, designada por *connectedMachinesThatWillContinueConnected*, aquelas que devem continuar conectadas, ou seja, aquelas que no novo esquema de distribuição não foram eliminadas. As restantes são armazenadas numa outra estrutura temporária, designada *connectedMachinesThatWillBeDisconnected*.
5. Alerta-se o utilizador caso existam canais cuja emissão irá ser terminada, ou máquinas cuja conexão irá ser fechada. Caso o utilizador aceite, prossegue-se para os próximos passos:
6. Terminam-se as emissões dos canais que estão referenciados na estrutura temporária *RunningChannelsThatWillBeClosed.*
7. Fecham-se as conexões das máquinas que estão referenciadas na estrutura temporária *connectedMachinesThatWillBeDisconnected.*
8. As *ArrayLists* de máquinas e canais do sistema geral do *LiveMeans* são esvaziadas.
9. São copiadas para a *ArrayList* de máquinas do *LiveMeans*, as máquinas do novo esquema de distribuição.
10. Para cada máquina copiada, verifica-se se esta está presente na estrutura temporária *connectedMachinesThatWillContinueConnected*. Em caso afirmativo, a estrutura e a *socket* é copiada, de forma a não se perder a conexão.
11. São copiadas para a *ArrayList* de canais do *LiveMeans*, os canais do novo sistema de distribuição, sendo também criadas as associações entre os canais e as máquinas responsáveis pela sua emissão.
12. Para cada canal copiado, verifica-se se este está presente na estrutura temporária *RunningChannelsThatWillContinueRunning*. Em caso afirmativo, copia-se a estrutura do canal, de forma a que se mantenha a emissão do mesmo.
13. Os dados são gravados na base de dados. Primeiro as tabelas *machine* e *channel* são esvaziadas, e de seguida preenchidas com os dados do novo esquema de distribuição.

6.8 Protocolo de comunicação

Para garantir a manutenção e escalabilidade do sistema, definiu-se um protocolo de comunicação. Este protocolo acentava inicialmente no uso de *CSV (Comma Separated Values)*, passando posteriormente a acentar no uso da sintaxe *XML*. As

vantagens e desvantagens de cada tecnologia, assim como os motivos que levaram à escolha do *XML* são discutidos no capítulo 3.4.

O formato das mensagens trocadas é semelhante em todos os casos, e consiste no seguinte:

<msg><op>OP_CODE</op> (...) parâmetros necessários (...) </msg>

O *OP_CODE* corresponde ao código da mensagem que está a ser enviada. A nível da aplicação, estes *opcodes* são associados a *strings* constantes de forma a facilitar a compreensão e utilização. Por exemplo, o *opcode* 23, representado pela *string alertAlreadyExistsError*, corresponde ao envio de uma mensagem de erro, por parte do servidor, em resposta a um pedido do cliente para a adição de um alerta relativo a uma determinada entidade, que informa o cliente que o alerta que ele está a tentar adicionar já se encontra adicionado. O anexo A deste relatório contém uma descrição de todas as mensagens trocadas entre o *back-end* e restantes aplicações.

Capítulo 7

Conclusões e Perspectivas de Trabalho Futuro

Nesta secção são apresentadas as conclusões sobre o trabalho desenvolvido durante o projecto, sendo abordadas as vantagens dos protótipos e o impacto que tiveram na empresa. A integração do projecto na empresa é também analisada juntamente com as perspectivas de trabalho futuro.

7.1 Avaliação dos resultados

Sendo que esta aplicação consiste num dos produtos de lançamento da empresa, a análise de viabilidade de cada tecnologia e o *benchmarking* dos protótipos foi sempre uma prioridade no desenvolvimento do projecto. Assim sendo, chegaram-se a algumas conclusões no que toca à *performance* da aplicação:

- Distribuindo um número de emissões por diversas máquinas, de forma a que cada máquina tenha, no máximo, cinco emissões *Live*, consegue-se uma boa fluidez na reprodução de todas as emissões nos clientes *Flash*. Se forem emitidos mais do que cinco canais na mesma máquina, os problemas começam a surgir, inicialmente visíveis numa quebra de fluidez, e, eventualmente, no *crash* do servidor de *RTMP*.
- No que toca ao número de utilizadores conectados, os testes elaborados não foram muito conclusivos. Fez-se uma experiência, com dez utilizadores, localizados fora da rede da empresa, a utilizar a aplicação e a visualizar as emissões *Live*. O sistema não apresentou problemas, apesar das emissões não se encontrarem fluídas do lado dos clientes. Tal facto prende-se com a limitada largura de banda existente na rede da empresa, em especial, da velocidade de *upload*. De resto, esta rede nunca foi pensada para suportar um sistema destes, sendo que o plano sempre foi mover o servidor para máquinas da *Amazon*, capazes de suportar elevados débidos de informação. De facto, o problema do número de canais é resolvido com o aumento de máquinas de *streaming*, enquanto que o aumento do número de utilizadores tem de obdecer forçosamente a um aumento da largura de banda.
- O módulo *Gateway* é um dos módulos mais interessantes da aplicação. Com efeito, o facto de os utilizadores poderem ser notificados remotamente via

telemóvel ou *e-mail*, acerca do início de um determinado programa ou da detecção de uma qualquer entidade, é bastante aliciante. Contudo, apenas faz sentido se a notificação for entregue num tempo útil. Em relação à notificação por *e-mail,* sabia-se à partida que não haveria qualquer problema, já que é entregue de forma quase imediata. Contudo, tendo em conta que os utilizadores não estão constantemente a utilizar o *e-mail*, esta questão da velocidade de notificação faz ainda mais sentido no caso das *sms*. No primeiro dia em que se utilizou a *gateway* de *sms* da *Shortcut,* os resultados não foram satisfatórios: Por vezes a *sms* demorava mais de quinze minutos a ser entregue ao utilizador. Contudo, após se contactar a empresa, esta informou que se tratava de um dia excepcional, em que decorria uma grande campanha publicitária de um cliente. De facto, desde esse dia que os resultados têm sido excelentes, com um período de tempo que varia entre os cinco e os quinze segundos desde que o conteúdo é detectado, até os utilizadores receberem a notificação nos seus dispositivos móveis.

- Outro caso em que a *performance* é fundamental prende-se com a comunicação entre o *front-end* e o *back-end*. No que toca aos alertas, tendo em conta que o anterior protótipo, desenvolvido em *Ruby*, acentava no *polling* constante ao servidor, por parte do cliente, tornava-se óbvio que os resultados seriam bastante superiores. De facto são, especialmente considerando que a fluidez da aplicação era bastante comprometida no primeiro protótipo, enquanto que neste deixou de ser uma preocupação. Mas não foi apenas na remoção deste *polling* que se obtiveram melhorias de *performance.* De facto, em toda a comunicação entre o cliente e o servidor, a nova forma de comunicação, ou seja, as *sockets* assíncronas, revelaram-se bastante mais eficientes que a utilização da tecnologia *WebOrb.* Esta melhoria de *performance* pode ser demonstrada pelo facto de no primeiro protótipo, quando o cliente fazia um pedido ao servidor, tal como a adição ou remoção de uma entidade da lista de alertas, surgir uma barra de *loading* enquanto essa informação era transmitida ao servidor, algo que no novo protótipo não acontece, apesar de a barra de *loading* existir, uma vez que o tempo de envio do pedido para o servidor é insignificante.
- Finalmente, no que toca à eficácia dos algoritmos para análise de som, pouco pode ser dito, já que durante a maior parte do desenvolvimento do projecto, todos os testes foram efectuados com simulações de algoritmos, tendo havido pouco contacto com os algoritmos propriamente ditos. De acordo com o que foi dito pela empresa, estes conseguem uma eficácia, por exemplo, no caso de reconhecimento de oradores, na ordem dos 80%. Em relação ao reconhecimento de faces, após alterações severas nas bibliotecas da *VeriLook* e a interpretação dos dados com heurísticas definidas, conseguiu-se uma eficácia na ordem dos 65%, no caso de existirem quatro entidades distintas. De notar que esta baixa eficácia pode subir acima dos 95%, caso os vídeos apresentem uma qualidade satisfatória e o indivíduo em questão esteja sempre em posição frontal.

7.2 Satisfação dos objectivos

Os objectivos pretendidos no início do projecto foram totalmente atingidos. De facto, esses objectivos iniciais prendiam-se apenas com o módulo *Video On Demand*. Contudo, após o término desse módulo, foi possível uma redefinição do projecto para englobar grande parte do sistema *LiveMeans*.

Também os objectivos que se pretendiam satisfazer, aquando da redefinição do projecto, foram completamente satisfeitos. Assim, foi construída uma aplicação, no modelo cliente-servidor, que permite que os utilizadores possam assistir a emissões televisivas em directo, ser notificados remotamente sempre que um determinado conteúdo em que estes estão interessados for detectado num dado canal, organizarem, segmentarem e partilharem uma colecção privada de audiovisuais de acordo com os seus conteúdos semânticos, entre outras possibilidades. O mais importante, sendo o factor que pretende resolver muitos problemas existentes na indústria de conteúdos, é o facto de toda a monitorização, análise e reconhecimento de conteúdos nas emissões televisivas e nas colecções de audiovisuais dos utilizadores, ser feita de forma inteiramente automática, sem recurso a mão de obra humana.

Para além do sistema *LiveMeans*, para notificação e reconhecimento automático de entidades em conteúdos audiovisuais, conseguiu-se ainda satisfazer outros objectivos secundários:

- Criação de três aplicações que, utilizadas em conjunto, permitem:
  - Criação de uma base de dados de conteúdos audiovisuais
  - Efectuar a segmentação manual desses conteúdos (*ground-truth)*
  - Criação de modelos de som e de faces para reconhecimento das entidades existentes nesses conteúdos.
- Investigação na área do reconheciment facial*,* tendo sido severamente adaptadas e modificadas, bibliotecas para reconhecimento de faces. Foram ainda realizados alguns protótipos, de raiz, para reconhecimento de faces. Esta investigação permitiu deixar em aberto o desenvolvimento de algoritmos de análise de vídeo para complementarem os algoritmos de análise de som, o que foi, desde o início, um dos objectivos secundários do projecto.

7.3 Integração na Empresa

A integração do produto na empresa foi imediata, estando a ser feitos os esforços necessários para instruir os outros funcionários da empresa no funcionamento das aplicações, em especial, do *back-end* do *LiveMeans*, para que este possa continuar a evoluir e a ser modificado. O *LiveMeans* é um dos produtos principais da *ClusterMedia Labs*, sendo, acima de tudo, uma forma de demonstrar o potencial dos algoritmos proprietários da empresa.

De facto, será muito em breve que este produto irá ser lançado oficialmente, bastando apenas alguns ajustes no *front*-end gráfico. Inicialmente irá ser lançado,

livremente, para o público em geral, sendo disponibilizado no domínio www.livemeans.com. Posteriormente, serão feitas versões específicas e adaptadas, de forma a integrar a tecnologia do *LiveMeans* nos sistemas de informação dos potenciais clientes.

7.4 Trabalho Futuro

Apesar de todo o sistema *LiveMeans* ser já bastante completo, existem ainda vários objectivos que seria interessante serem conseguidos em trabalho futuro:

- Criação de modelos de reconhecimento para um grande número de entidades.
- Inclusão, na análise dos canais *Live*, de algoritmos de análise de imagem, quer para reconhecimento de faces, quer para reconhecimento do símbolo de um canal, de forma a saber se este está a transmitir um programa ou publicidade.
- Melhorar a robustez do servidor de *RTMP*, o *Red5*, e integrar-lhe as funcionalidades do *vlc* e do *ffmpeg*, de forma a serem necessários menos recursos por cada emissão televisiva.
- Melhoramento da escalabilidade do sistema, em especial com recurso a técnicas de *sharding*, ou seja, divisão da base de dados, replicação de dados, utilização de várias máquinas para o *back-end,* etc.
- Criação de *schemas* para validação das mensagens *XML* trocadas entre os diferentes módulos do sistema.
- Possibilidade de enviar uma *mms*, em vez de uma *sms*, no caso da notificação remota, possibilitando a visualização de um excerto da entidade detectada ou apenas de uma imagem representativa da mesma.
- Desenvolvimento de uma solução para definir o final de uma detecção – Os modelos existentes permitem a detecção do início de alguma entidade, mas não o seu final. Para efeitos de gravação dos conteúdos, seria interessante o desenvolvimento de uma solução ou tecnologia que permitisse detectar o final da detecção de uma determinada entidade. Actualmente, sempre que uma entidade é reconhecida, o canal em que esta foi detectada é gravado durante um determinado número de minutos. No entanto, seria importante a utilização de alguma tecnologia mais avançada. Tal tecnologia, no caso de oradores, seria fácil de implementar, bastando, registar o final da detecção assim que o *confidence* de detecção da voz da entidade baixasse significativamente durante um determinado período de tempo. Já no caso da detecção de músicas, seria interessante a detecção de mudança de uma música para outra. Uma situação mais complicada prende-se com um programa de televisão. Sendo que este é detectado pelo *jingle* inicial que o caracteriza, a detecção do seu final é mais complicada. Ainda assim, a detecção do final de um programa poderia ser conseguida de várias formas:
    - Através de um eventual *jingle* de fim de programa.
    - Pela detecção de um outro programa.
    - Pela consulta de uma tabela de programação de apoio.

- Pela passagem da emissão para publicidade, detectada pelo desaparecimento do logótipo do canal televisivo, ou outras alterações do sinal audiovisual.

Referências

[ADI08] Agência de Inovação. Página da instituição, Julho 2008. http://www.adi.pt/

[BLE08] The University of Texas at Austin. In Memoriam: Woodrow W. Bledsoe, Julho 2008. http://www.utexas.edu/faculty/council/1998-1999/memorials/Bledsoe/bledsoe.html

[BRE08] Break. Página do portal, Julho 2008. http://www.break.com

[CC08] Cord, Matthieu and Cunningham, Pádraig (Eds.). *Machine Learning Techniques for Multimedia.* s.l. : Springer, 2008.

[CLU08] ClusterMedia Labs. Página da empresa, Julho 2008. http://www.clustermedialabs.com

[DLB08] Duke Listens Blog. What's on your iPod?, Jilho 2008. http://blogs.sun.com/plamere/entry/what_s_on_your_ipod

[DM08] Daily Motion. Página do portal, Julho 2008. http://dailymotion.com

[FEU08] Faculdade de Engenharia da Universidade do Porto. Página do Mestrado Integrado em Engenharia Informática e Computação, Julho 2008. http://www.fe.up.pt/si/cursos_geral.FormView?P_CUR_SIGLA=MIEIC

[FOR08] Fortune. YouTube looks for the money clip, Julho 2008. http://techland.blogs.fortune.cnn.com/2008/03/25/youtube-looks-for-the-money-clip/

[FR08] College of Engineering, Purdue University. Face Recognition, Julho 2008. https://engineering.purdue.edu/people/mireille.boutin.1/ECE301kiwi/FaceRecognition

[FRS08] Wikipedia. Facial recognition system, Julho 2008. http://en.wikipedia.org/wiki/Facial_recognition_system

[HSW08] How Stuff Works. Facial recognition, Julho 2008. http://electronics.howstuffworks.com/facial-recognition1.htm

[KB95] Khoshafian, Setrag and Baker, Brad. *Multimedia and Imaging Databases.* s.l. : Morgan Kaufmann, 1995.

[LAS08] Last.Fm. Página do portal, Julho 2008. http://www.last.fm/

[LM08] LiveMeans. Página do produto, Julho 2008. http://www.livemeans.com

[MAL04] Maltoni, Davide. *Biometric Authentication: ECCV 2004 International Workshop.* s.l. : Springer, 2004.

[MC08] MetaCafe. Página do portal, Julho 2008. http://www.metacafe.com/

[NT08] Neurotechnology. Página da empresa, Julho 2008. http://www.neurotechnology.com/

[PAN08] Pandora. Página do portal, Julho 2008. http://www.pandora.com

[SHO08] Shortcut. Página da empresa, Julho 2008. http://www.shortcut.pt/

[SR08] Center for Spoken Language Understanding. Speaker Recognition, Julho 2008. http://www.cslu.ogi.edu/HLTsurvey/ch1node9.html

[SV08] Sapo Vídeos. Página do portal, Julho 2008. http://videos.sapo.pt

[TAY08] Taylor, Paul. *Text-to-Speech Synthesis.* s.l. : Cambridge University Press, 2008.

[TG08] TeleGraph. Web could collapse as video demand, Julho 2008. http://www.telegraph.co.uk/news/uknews/1584230/Web-could-collapse-as-video-demand-soars.html

[TSL08] The Script Library. PHP script WebVideo Grabber, Julho 2008. http://www.thescriptlibrary.com/Default.asp?Action=Display&Level=Category3&ScriptLanguage=PHP&Category1=Other&Category2=Other&Title=WebVideo%20Grabber

[TVN08] Wikipedia. List of United States over-the-air television networks, Julho 2008. http://en.wikipedia.org/wiki/List_of_United_States_over-the-air_television_networks

[USA08] Usa Today. YouTube serves up 100 million videos a day online, Julho 2008. http://www.usatoday.com/tech/news/2006-07-16-youtube-views_x.htm

[WEC06] Wechsler, Harry. *Reliable Face Recognition Methods: System Design, Implementation and Evaluation* . s.l. : Springer, 2006.

[WII04] Wiil, Uffe Kock. *Computer Music Modeling and Retrieval.* s.l. : Springer, 2004.

[YT08] YouTube. Página do portal, Julho 2008. http://www.youtube.com

[YTA08] YouTube. Lista de todos os vídeos, Julho 2008. http://www.youtube.com/results?search_query=*

Carlson, Lucas and Richardson, Leonard. *Ruby Cokbook*. O'Reilly, 2006.

Dev Articles. Building XML Web Services Using C# and ASP.NET, Julho 2008. http://www.devarticles.com/c/a/ASP.NET/Building-XML-Web-Services-Using-C-sharp-and-ASP.NET/

Devhood. Asynchronous server socket using C#, Julho 2008. http://www.devhood.com/Tutorials/tutorial_details.aspx?tutorial_id=709

Electroteque. Red5 Documentation, Julho 2008. http://www.electroteque.org/docbook/red5.pdf

Microsoft. Microsoft Developer Network, Julho 2008. http://msdn.microsoft.com/pt-pt/default(en-us).aspx

René Nyffenegger's collection of things on the web. C++ Socket Class for Windows, Julho 2008. http://www.adp-gmbh.ch/win/misc/sockets.html

Research and Instruction in Technologies for Electronic Security. Socket Programming in C/C++, Julho 2008. http://www.rites.uic.edu/~solworth/sockets.pdf

TagusPark. Tutorial de OpenCV para Tótós, Julho 2008. http://web.tagus.ist.utl.pt/~alexandra.ribeiro/tutorialopencv.html

Troelsen, Andrew. *Pro C# 2008 and .NET 3.5 Platform*. Apress, 2008.

Anexo A

Protocolo de Comunicação

Este anexo apresenta, de forma detalhada, todas as mensagens trocadas entre o *back-end* e as restantes aplicações do sistema *LiveMeans*.

- **Comunicações efectuadas do *Back-end* para o Cliente *Flash*:**
  - Send Ping
    - *Op Code*: 00
    - Parâmetros: Não tem
    - Objectivo: Enviar um *Ping* ao cliente para verificar se a conexão continua activa.
  - Ask for Login Confirmation
    - *Op Code*: 26
    - Parâmetros: Um conjunto aleatório de caracteres (*salt*)
    - Objectivo: O cliente passou a primeira fase de autenticação. Para completar o procedimento de *login* deve enviar uma *hash MD5* gerada a partir da concatenação do código *salt* com a *password* do cliente.
  - Login Accepted
    - *Op Code*: 02
    - Parâmetros: Não tem
    - Objectivo: A segunda fase de *login* foi validada e o servidor envia uma mensagem a informar o cliente que já se encontra autenticado.
  - Login Rejected
    - *Op Code*: 03
    - Parâmetros: Não tem

- Objectivo: O *login* não foi aceite por isso o cliente ainda não se encontra autenticado.

- Send Entities List
  - *Op Code*: 08
  - Parâmetros: O *id,* nome, tipo e restantes parâmetros específicos daquele tipo de entidade, por exemplo, profissão, no caso de oradores.
  - Objectivo: Aquando do *login*, é enviada uma lista de todas as entidades existentes no sistema.
- Send Entities Error
  - *Op Code*: 09
  - Parâmetros: Não tem
  - Objectivo: Enviado quando ocorre uma falha no envio da lista de entidades para o cliente.
- Send Personal Alerts
  - *Op Code*: 19
  - Parâmetros: *ids* das entidades nas quais o cliente está interessado.
  - Objectivo: Aquando do *login*, é enviada uma lista de todas as entidades nas quais o utilizador já se encontra interessado.
- Send Personal Alerts error
  - *Op Code*: 20
  - Parâmetros: Não tem
  - Objectivo: Enviado quando ocorre uma falha no envio da lista das entidades nas quais o utilizador está interessado.
- Send Channels
  - *Op Code*: 25
  - Parâmetros: *id*, nome, *ip* e porto de acesso a cada emissão *Live*.
  - Objectivo: Aquando do *login*, enviar uma lista de todos os canais *live* disponíveis e informação para o cliente aceder a eles e os reproduzir na *interface*.
- Add Alert Successful
  - *Op Code*: 11
  - Parâmetros: *id* da entidade que foi adicionada à lista de alertas.
  - Objectivo: Enviado em resposta a um pedido do cliente para adição de uma entidade à sua lista de alertas. O cliente é informado que houve sucesso na operação.

- Add Alert Error
  - *Op Code*: 12
  - Parâmetros: *id* da entidade que o cliente pretendia adicionar à sua lista de alertas.
  - Objectivo: Enviado em resposta a um pedido do cliente para adição de uma entidade à sua lista de alertas. O cliente é informado que a operação falhou por um motivo desconhecido.
- Alert Already Exists Error
  - *Op Code*: 23
  - Parâmetros: *id* da entidade que o cliente pretendia adicionar à sua lista de alertas.
  - Objectivo: Enviado em resposta a um pedido do cliente para adição de uma entidade à sua lista de alertas. O cliente é informado que a operação falhou, uma vez que a entidade que está a tentar adicionar, já se encontra na sua lista de alertas.
- Remove Alert Successful
  - *Op Code*: 21
  - Parâmetros: *id* da entidade que o utilizador removeu da sua lista de alertas.
  - Objectivo: Enviado em resposta a um pedido do cliente para remoção de uma entidade da sua lista de alertas. O cliente é informado que a operação foi concluída com sucesso.
- Remove Alert Failed
  - *Op Code*: 22
  - Parâmetros: *id* da entidade que o utilizador pretendia remover da sua lista de alertas.
  - Objectivo: Enviado em resposta a um pedido do cliente para remoção de uma entidade da sua lista de alertas. O cliente é informado que a operação falhou por um motivo desconhecido.
- Alert Does Not Exist Error
  - *Op Code*: 25
  - Parâmetros: *id* da entidade que o utilizador pretendia remover da sua lista de alertas.
  - Objectivo: Enviado em resposta a um pedido do cliente para remoção de uma entidade da sua lista de alertas. O cliente é informado que a operação falhou, uma vez que a entidade que este pretendia remover, não se encontra na sua lista de alertas.
- Edit Info Successful

- *Op Code*: 14
- Parâmetros: Não tem.
- Objectivo: O cliente é notificado que as modificações efectuadas ao seu perfil foram gravadas com sucesso.

o Edit Info Error

- *Op Code*: 15
- Parâmetros: Não tem.
- Objectivo: O cliente é notificado que a operação de gravação das suas modificações ao seu perfil falhou por um motivo desconhecido.

o Not Logged Error

- *Op Code*: 17
- Parâmetros: Não tem.
- Objectivo: Enviado sempre que um utilizador não autenticado tenta efectuar operações para além daquelas envolvidas no processo de *login*.

o Error Accessing Database

- *Op Code*: 18
- Parâmetros: Não tem.
- Objectivo: Enviado para o cliente sempre que, na satisfação de um seu pedido, haja um erro no acesso à base de dados.

o Analysis Results

- *Op Code*: 33
- Parâmetros: Nome do ficheiro que foi transferido, código gerado pelo cliente aquando do envio para identificação e validação, e informação acerca das regiões segmentadas do vídeo. Para cada região, é enviado o *inPoint* e *OutPoint* assim como o *id* da entidade reconhecida nessa região e o nível de *confidence* com que a mesma foi reconhecida. É ainda enviado o *id* da entidade mais proeminente durante todo o vídeo.
- Objectivo: Resposta ao envio de um vídeo por parte de um utilizador, com o resultado da análise e segmentação efectuada pelos algoritmos de análise de voz e de faces.

o Send Alert

- *Op Code*: 33
- Parâmetros: *id* da entidade que foi reconhecida e *id* do canal em que esta foi detectada.

- Objectivo: Quando uma das entidades em que o utilizador está interessado, é detectada, o servidor envia esta mensagem a informar acerca da sua detecção assim como o canal em que a detecção sucedeu. O *front-end* gráfico apresenta então, de forma visual, essa detecção.

- **Comunicações efectuadas do cliente *Flash* para o *Back-End:***
  - Send Credentials
    - *Op Code*: 01
    - Parâmetros: nome de utilizador e *hash MD5* da *password*.
    - Objectivo: Iniciar o processo de *login.*
  - Send Login Confirmation
    - *Op Code*: 27
    - Parâmetros: *Hash MD5* da concatenação do código *salt* enviado pelo utilizador com a palavra-chave.
    - Objectivo: Concluir o processo de *login,* provando ao servidor que, para além da *Hash MD5*, a própria palavra-chave também é conhecida, sem que esta seja enviada através da rede.
  - Add Alert
    - *Op Code*: 10
    - Parâmetros: *id* da entidade que se pretende adicionar à lista de alertas.
    - Objectivo: Adicionar uma determinada entidade à lista de alertas.
  - Remove Alert
    - *Op Code*: 20
    - Parâmetros: *id* da entidade que se pretende remover da lista de alertas.
    - Objectivo: Remover uma determinada entidade da lista de alertas.
  - Edit Info
    - *Op Code*: 13
    - Parâmetros: dados pessoais que se pretendem modificar, tais como o nome, o *e-mail,* as preferências em relação a notificação via *sms* e/ou *e-mail,* etc.
    - Objectivo: Efectuar alterações no perfil do utilizador.
  - Upload Complete
    - *Op Code*: 32

- Parâmetros: Nome do ficheiro acabado de transferir e código aleatório gerado pelo cliente para identificação e autenticação do ficheiro.
- Objectivo: Notificação do servidor acerca do final da transferência de um ficheiro para o *back-end*, para que este o analise com os algoritmos da empresa.

- *Log Out*
  - *Op Code*: 99
  - Parâmetros: Não tem.
  - Objectivo: O utilizador pretende abandonar a sessão.

- **Comunicações efectuadas do *Back-End* para o *Atomic Remote Process Manager* de uma *Broadcast Machine:***
  - *Launch Channel*
    - *Op Code*: 28
    - Parâmetros: Parâmetros do *vlc* e *ffmpeg* adequados para a captura e reenvio de uma emissão *Live.* É ainda enviado um *id* único que identifica a emissão.
    - Objectivo: O servidor pretende iniciar a emissão de um dado canal, numa determinada máquina de *broadcast,* enviando para ela os dados necessários para invocar os programas responsáveis por essa emissão.
  - *Close Channel*
    - *Op Code*: 31
    - Parâmetros: *id* que identifica a emissão.
    - Objectivo: O utilizador pretende terminar a emissão de um dado canal, que está a ser emitido numa determinada máquina de *broadcast*, enviando para ela o *id* da emissão para que esta possa terminar os processos envolvidos na emissão do canal.

- **Comunicações efectuadas do *Atomic Remote Process Manager* de uma *Broadcast Machine* para o *Back-End:***
  - *Channel Closed*
    - *Op Code*: 29
    - Parâmetros: *id* que identifica a emissão.
    - Objectivo: Uma dada emissão foi terminada de forma inesperada, por exemplo, porque os processos falharam ou porque a emissão foi interrompida. Assim sendo, o servidor é notificado do

acontecimento de forma a fazer as alterações necessárias nas estruturas das emissões, notificar graficamente os administradores do *back-end,* e permitir a retransmissão do canal.

- **Comunicações efectuadas de um *Audio Engine* para o *Back-End:***
  - *Audio Engine Sending Alert*
    - *Op Code*: 34
    - Parâmetros: *id* da entidade reconhecida e *id* do canal em que esta foi detectada.
    - Objectivo: É desta forma que um determinado *Audio Engine,* após a detecção de uma qualquer entidade num dado canal *Live*, notifica o *back-end* acerca desse reconhecimento.