Universidade do Minho
Escola de Engenharia

João Pedro Domingues Cadinha

Interpretador SMIL de Alta Performance para Controlo de Apresentações Multimédia num Servidor de Streaming de Media para Dispositivos Móveis.

Dezembro de 2009

Universidade do Minho
Escola de Engenharia

João Pedro Domingues Cadinha

Interpretador SMIL de Alta Performance para Controlo de Apresentações Multimédia num Servidor de Streaming de Media para Dispositivos Móveis.

Dissertação de Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação do
Professor Doutor Alberto Manuel Brandão Simões

Dezembro de 2009

Agradecimentos

Aos meus pais, irmão e família que desde cedo me incentivaram e ajudaram em tudo.

À Vanessa que desde o meu primeiro dia na Portugal Telecom Inovação me apoiou, deu força e "mostrou o caminho".

A toda a equipa do DRP4 e do pólo do Porto pelo excelente ambiente de trabalho.

Ao Alberto pela paciência que demonstrou comigo, em especial nos últimos dias antes da entrega desta dissertação.

Ao Barreira pelas longas conversas ao almoço, sucessivos incentivos e críticas construtivas.

A todos os meus amigos.

Dissertação financiada pela Portugal Telecom Inovação.

Título

Interpretador SMIL de Alta Performance para Controlo de Apresentações Multimédia num Servidor de Streaming de Media para Dispositivos Móveis.

Resumo

O Synchronized Multimedia Integration Language (SMIL) é um padrão definido pelo World Wide Web Consortium (W3C), baseado na eXtended Markup Language (XML), usada no controlo de apresentações multimédia. Esta linguagem é usada principalmente nos Serviços de Mensagens Multimédia mas, actualmente, também é usada nos High Definition DVD para interactividade e para vídeos na Internet.

Este documento descreve de que forma o Ambulant Player, um animador de SMIL open-source, foi modificado para controlar apresentação multimédia fornecidas por um *Media Server* para dispositivos móveis. A implementação modular do Ambulant baseada em *code factories* permitiu que o seu módulo de visualização fosse substituído por um mais simples. Este novo módulo envia mensagens para um *Media Server* em vez de reproduzir os elementos de média. Usando o Ambulant Player como base para este interpretador tornou o desenvolvimento mais rápido e permitiu obter uma ferramenta que respeite a recomendação do W3C de uma forma simples.

Como resultado final obteve-se um interpretador que demonstrou ser robusto, suportando cento e oitenta sessões concorrentes e servindo cerca de sessenta mil sessões sem erros.

Title

High Performance SMIL Interpreter for Media Presentation Control in a Media Streaming Server for Mobile Devices.

Abstract

The Synchronized Multimedia Integration Language (SMIL) is a World Wide Web Consortium (W3C) standard language based on the eXtended Markupe Language (XML), used to control multimedia presentations. This language was mainly used in Multimedia Messaging Services but nowadays is also being used in High Definition DVDs for advanced interactivity and in Internet Video.

This document describes how Ambulant Player, a pure open-source SMIL player, was modified to control the media presentation provided by a Media Streaming Server for mobile devices. Ambulant's factory system allowed to replace the basic renderer by a simpler one. This new renderer sends messages to the Media Server instead of playing the media. Using Ambulant as a base for this interpreter made the development faster and made it easier to obtain a W3C compliant tool.

The interpreter obtained as final result of this project showed to be robust, supporting one hundred and eighty concurrent sessions and serving about sixty thousand sessions without errors.

Índice

Índice de Figuras

Índice de Tabelas

Terminologia

Animated GIF – Animated Graphics Interchange Format.

API – Application Programmer Interface.

ASCII – American Standard Code for Information Interchange.

Billing – Facturação. Termo usado de forma generalizada na área das telecomunicações.

Browser – Navegador. Termo usado correntemente que desambigua a sua tradução.

Call Centers – Centro de Suporte. Dado ser a designação vulgarmente usada optou-se por não traduzir o termo

Content Management Platform – Plataforma de Gestão de Conteúdos. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

Content Management Server – Servidor de Gestão de Conteúdos. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

Content Manager – Gestor de Conteúdos. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

CSMIL – Interpretador SMIL.

DSP – Digital Signal Processor.

DVD – Digital Versatile Disc

Flag – Parâmetro usado na compilação. Dado ser a designação vulgarmente usada optou-se por não traduzir o termo.

FTP – File Transfer Protocol.

GUI – Graphical User Interface.

HTML – Hyper Text Markup Language.

HTTP – Hypertext Transfer Protocol.

IMS – IP Multimedia Subsystem.

InoSmil – Interpretador SMIL.

IP – Internet Protocol Address

JPG – Joint Photographic Experts Group.

LCM – Lightweight Communications and Marshalling.

LGPL – Lesser General Public License.

Live Feed Distributer – Distribuidor de conteúdos ao vivo. Dado ser a designação vulgarmente usada optou-se por não traduzir o termo.

Live feeds – Conteúdos ao vivo. Dado ser a designação vulgarmente usada optou-se por não traduzir o termo.

Logo Insertion – Inserção de logótipos. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

Media Server – Servidor de Média. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

Mid-roll clips – Vídeos, normalmente publicitários, reproduzidos a meio de uma sessão de vídeo. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

MMS – Multimedia Messaging Service.

Mobile TV – Televisão para dispositivos móveis. Dado ser a designação vulgarmente usada optou-se por não traduzir o termo.

MPEG – Moving Picture Experts Group.

MRF – Media Resource Function.

Open-source – Código fonte acessível. Dado ser a designação vulgarmente usada optou-se por não traduzir o termo.

Picture-in-picture – Sobreposição de duas imagens ou dois vídeos, um dentro do outro. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

Player – Aplicação que reproduz graficamente conteúdos multimédia. Dada a inexistência de uma tradução comum optou-se por manter o termo.

Playlist – Lista de reprodução. Dado ser a designação vulgarmente usada optou-se por não traduzir o termo.

PNG – Portable Network Graphics.

Pos-roll clips – Vídeos, normalmente publicitários, reproduzidos após uma sessão de vídeo. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

Pre-roll clips – Vídeos, normalmente publicitários, reproduzidos no iníciode uma sessão de vídeo. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

Render – Construção gráfica de um documento. Dada a inexistência de uma tradução comum optou-se por manter o termo.

Renderer – Módulo responsável pela construção gráfica de um documento. Dada a inexistência de uma tradução comum optou-se por manter o termo.

RTP – Real-time Transport Protocol.

RTSP – Real Time Streaming Protocol.

SIP – Session Initiation Protocol.

SMIL – Synchronized Multimedia Integration Language.

State-of-the-art – Estado de arte. Dado ser a designação vulgarmente usada optou-se por não traduzir o termo.

SVG – Scalable Vector Graphics.

Team Leader – Líder de equipa. Dado ser a designação vulgarmente usada num contexto empresarial optou-se por não traduzir o termo.

Technical Product Manager – Gestor Técnico do Produto. Dado ser a designação vulgarmente usada num contexto empresarial optou-se por não traduzir o termo.

Text-overlay – Sobreposição de texto numa apresentação multimédia. Como esta designação é normalmente usada num contexto comercial é mais corrente o seu uso em Inglês.

UMSP – Unified Messaging Service Providers.

URL – Uniform Resource Locator.

VoiceXML – Norma do W3C usada para especificar diálogos de voz interactivos entre um ser humano e um computador.

W3C – World Wide Web Consortium.

WSS – Windless Streaming Server.

XML – Extensible Markup Language.

1. Introdução

A rápida evolução dos dispositivos móveis, como telemóveis e *smart phones* entre outros, levou à criação de novos serviços por parte das operadoras móveis, de forma a acompanhar a evolução destes dispositivos. O número de utilizadores a usarem serviços de vídeo chamada ou *mobile TV* aumentou consideravelmente, havendo uma grande aposta do mercado nesta área, criando novos serviços e disponibilizando novas funcionalidades, o que levou a um aumento substancial da qualidade de serviço. Algumas funcionalidades, como por exemplo o *text-overlay* ou o *Picture-in-picture*, são implementadas nos seus servidores de *streaming* de média e as definições das apresentações de média são guardadas numa *playlist* fornecida por um servidor de conteúdos. Estas *playlists* controlam a apresentação multimédia (posicionamento, *timing*, entre outros), sendo o SMIL[1] a escolha óbvia para a sua representação, pois é um standard do W3C[3] e quase não tem limites no controlo de apresentações de media.

1.1 Enquadramento

A Portugal Telecom Inovação (PT Inovação) foi constituída em Maio de 1999 e tem como actividade principal garantir o processo de inovação da Portugal Telecom, SA e empresas participadas (Grupo Portugal Telecom) através da prestação de serviços às mesmas. A PT Inovação é a âncora tecnológica do Grupo Portugal Telecom. O centro de excelência deste, com mais de meio século de história, constituindo um valor estratégico para as telecomunicações portuguesas.

A Direcção de Desenvolvimento de Redes e Protocolos é, na PT Inovação, responsável por todo o ciclo de desenvolvimento de produtos integráveis nas redes de telecomunicações ao nível de plataformas de serviço e *core network*. Os domínios de actuação cobertos vão desde soluções para redes de dados, passando por soluções de *messaging*, localização geográfica

e celular, soluções multimédia, redes inteligentes e culminando no desenvolvimento de produtos para redes de próxima geração.

A PT Inovação apresenta actualmente uma gama de produtos aplicados a redes multimédia, com maior evidência para o Windless Streaming Server (WSS) e o ip-Windless que pode funcionar como *standalone* Media Server para redes IP ou desempenhar o papel de Media Resource Function (MRF) num ambiente IP Multimedia Subsystem (IMS).

Ambos os produtos disponibilizam conteúdos multimédia na camada de rede e efectuam diversos tipos de processamento avançado sobre estes mesmos conteúdos.

O WSS é uma *gateway* para *streaming* de conteúdos ao vivo ou a pedido para utilizadores na internet ou redes móveis, podendo funcionar como um RTSP *proxy* ou *server* disponibilizando conteúdos *offline* ou *live feeds*, em ambientes *mobile TV, mobile advertisement* e outro tipo de clientes RTSP. O ip-Windless é uma plataforma multi-serviços destinada a aplicações interactivas de voz e vídeo, reprodução e gravação de anúncios, audioconferência e futuramente também vídeo-conferência. O ip-Windless pode ser utilizado em *Call Centers*, *Unified Messaging Service Providers*, fornecedores de serviços de conferência, serviços de telecomunicações e de internet.

Com este projecto pretendeu-se desenvolver um interpretador para o *standard* Synchronized Multimedia Integration Language (SMIL) do W3C e integrá-lo nos produtos WSS e ip-Windless de forma a usar algumas particularidades desta linguagem, aplicando-as a serviços como *Mobile Advertisement*, *Picture-in-picture* em *streams* multimédia, Vídeo Portal e Vídeo-Conferência. A introdução de um interpretador SMIL nestes dois produtos da PT Inovação simplifica as tarefas a desempenhar por um *Content Management Server*, permitindo um fácil controlo espacial e temporal dos recursos multimédia disponibilizados por estas soluções, conforme exemplificado na figura 1.

O desenvolvimento do interpretador de SMIL, e a título de exemplo, facilitada a especificação de parâmetros que permitem ao WSS a manipulação de *streams* multimédia em tempo real sobrepondo-lhes *Text-Overlay*, *Logo Insertion, Animated* GIFs, *pre/mid/end-roll clips* e especificar no ip-Windless a

organização do mosaico com o vídeo dos vários participantes de uma vídeo-conferência.

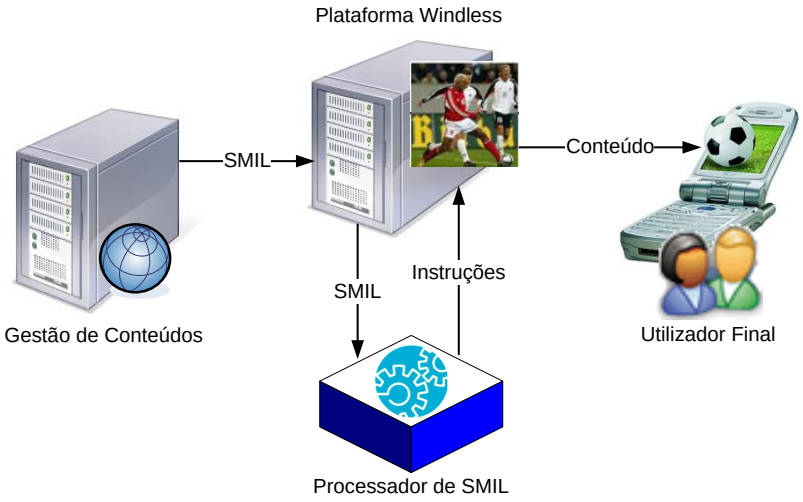

Figura 1 - Enquadramento do interpretador SMIL na plataforma Windless

Este projecto foi realizado no pólo do Porto da PT Inovação, na equipa DRP4 onde, em simultâneo com o desenvolvimento deste, foram realizadas tarefas de formação, nomeadamente sobre produtos da empresa. No que diz respeito ao acompanhamento, este foi realizado de uma forma mais próxima pelo *Team Leader* da equipa sendo que, fundamentalmente na componente de investigação, este acompanhamento foi reforçado pelos *Technical Product Managers*, que guiaram os trabalhos por forma a terem o necessário enquadramento nas actividades da Direcção. Deste modo todo o trabalho realizado é essencialmente prático.

1.2 Objectivos

O trabalho executado organizou-se em duas componentes distintas. A componente inicial de investigação, que visava a obtenção de conhecimentos sobre a realidade da PT Inovação e sobre o domínio específico do estágio, mediante investigação quer das normas existentes quer das tendências tecnológicas que permitem essas implementações. Como resultado desta componente inicial foi produzido um relatório *state-of-the-art* que serviu de base para esta dissertação. A componente complementar, de desenvolvimento,

visava a aplicação do conhecimento obtido na elaboração de um protótipo de um potencial no domínio de actividade, o qual foi desenvolvido segundo o processo de desenvolvimento na PT Inovação. Desta forma, os objectivos gerais deste projecto foram:

1. Estudo e elaboração de um relatório com o estado de arte neste domínio dos interpretadores e visualizadores/*players* SMIL na área do open-source.
2. Desenvolvimento de um protótipo que realize a interpretação de documentos na linguagem SMIL, cingindo-se às funcionalidades multimédia existentes no WSS e ip-Windless da PT Inovação.
3. Implementação das interfaces necessárias para a utilização dos recursos multimédia do WSS através da linguagem SMIL.
4. Implementação das interfaces necessárias para o controlo espacial da funcionalidade de vídeo-conferência no ip-Windless via API desta plataforma, através da linguagem SMIL.

1.3 Métodos

Para que se atinjam resultados de elevada qualidade, este trabalho foi planeado em duas etapas, iniciando-se com uma investigação e experimentação que visava obter e aprofundar conhecimentos sobre SMIL e alguns dos seus visualizadores, que resultou num relatório com o estado de arte neste domínio. Concluída esta etapa, iniciou-se a especificação e o desenho do protótipo do produto, utilizando os conhecimentos adquiridos na etapa anterior, seguindo-se do desenvolvimento e testes do produto.

O produto final deverá apresentar elevada qualidade e um desempenho optimizado, uma vez que é um produto comercial e terá de ser capaz de responder rapidamente a um elevado número de pedidos. Desta forma, o resultado terá que apresentar uma qualidade elevada e os testes ao produto tiveram um papel fundamental, ou seja, foi utilizada uma metodologia qualitativa para que se pudesse obter um interpretador muito eficiente.

1.4 Estrutura do Documento

Este documento, numa primeira fase, explica o seu enquadramento, contexto e as metodologias usadas para a concepção deste projecto. De seguida é feito um pequeno enquadramento da linguagem SMIL e explicada de que forma esta é útil para o cumprimento dos objectivos propostos neste projecto. Após esta introdução à linguagem é feita uma análise tecnológica de *players* de SMIL encontrados, justificando-se a razão do uso do Ambulant Player[4] como guia para o interpretador (CSMIL). No capítulo 4 é aprofundada a arquitectura deste *player*, passando-se pelos seus princípios de desenvolvimento, bibliotecas de terceiros usadas e uma análise da sua arquitectura. Esta primeira parte do documento corresponde a um estudo do estado de arte sobre o tema desta dissertação.

Numa segunda parte, correspondente a um trabalho mais prático, temos o capítulo 5, onde se descreve o desenvolvimento do interpretador, o capítulo 6, onde se mostra os resultados obtidos nos testes efectuados a este, e o capítulo 7, onde se refere as conclusões e o trabalho futuro.

2. SMIL

A linguagem SMIL é um *standard* para coreografar apresentações multimédia onde áudio, vídeo, texto e gráficos são combinados em tempo real. Esta linguagem permite controlar o início, fim, duração, tamanho e outros atributos dos seus elementos de média. O SMIL também permite que os elementos multimédia definidos nos seus documentos sejam reproduzidos sequencialmente ou paralelamente. Esta linguagem encontra-se escrita em XML e apresenta algumas similaridades com o HTML. As suas características tornam esta linguagem ideal para o controlo de apresentações multimédia, o que explica o facto de esta estar a adquirir cada vez maior importância hoje em dia, daí o ser usada em mensagens multimédia, *web video players* ou aplicações de leitura de livros[2].

O SMIL tornou-se uma recomendação do W3C em Junho de 1998 (versão 1.0) e actualmente encontra-se na sua versão 3.0, que passou a versão recomendada pelo W3C em Dezembro de 2008. Esta linguagem possui uma estrutura modular que permite que os utilizadores a combinem com outras linguagens baseadas em XML, como o SVG ou o VoiceXML. O SMIL ainda não é reconhecido de forma nativa pelos browsers, sendo necessária instalação de *plug-ins* para que estes suportem a linguagem. No entanto, devido ao aumento constante do seu número de utilizadores e uma vez que esta linguagem adquire cada vez maior importância no contexto Web, os browsers estão lentamente a começar a incorporar as funcionalidades desta linguagem.

2.1 Documentos SMIL

Tal como foi dito anteriormente, o SMIL apresenta algumas semelhanças com o HTML. Assim, tal como nos documentos escritos em HTML, os documentos SMIL apresentam duas secções, uma secção **head** e uma secção **body**. A secção **head** é opcional e contém informações sobre o *layout* do documento e metadata. A secção **body** e contém informações sobre o *timing* e sobre os elementos multimédia que serão reproduzidos. O *timing* dos

elementos multimédia desta secção são controlados principalmente pelas etiquetas **par** (paralelo) e **seq** (sequência). Para além destas etiquetas, esta linguagem também permite associar propriedades aos seus elementos que permitem controlar os seus *timings* e também algumas das suas propriedades.

Os elementos de media nesta linguagem são referidos através de URLs o que permite que estes estejam definidos em múltiplos servidores. Uma outra característica desta linguagem é que permite associar larguras de banda diferentes para cada elemento de media. No que diz respeito às extensões dos ficheiros escritos nesta linguagem, estes podem apresentar a extensão **smi** ou **smil**, sendo esta última a mais comum.

2.1.1 Definição de *Layouts*

Tal como foi dito anteriormente, os documentos SMIL são compostos por duas partes, sendo a primeira responsável pela definição do *layout* do documento. Esta parte é definida dentro da secção **head** e é opcional. Sempre que a secção **head** é definida é necessário definir um **root-layout**, que irá representar a configuração da janela onde será reproduzido o documento. Novas regiões podem ser definidas dentro desta através do elemento **region**. Os elementos **region** e **root-layout** partilham os mesmos atributos, que se encontram na tabela 1.

Apesar de apenas ser possível definir uma região do tipo root-layout, não existe limite para o número de elementos region por documento. A definição das regiões do documento é importante uma vez que posteriormente é necessário indicar a região onde os elementos multimédia são reproduzidos. No contexto em que se pretende desenvolver este interpretador, a definição de regiões é importante, uma vez que é através do uso de duas regiões que é possível implementar funcionalidades como o *picture-in-picture*.

Atributo	Tipo de Dados	Descrição
Id	Texto	Representa o identificador da região.
Width	Inteiro	Define a largura da região.
Height	Inteiro	Define a altura da região.
Top	Inteiro	Define a distância da região à margem superior da janela.
Bottom	Inteiro	Define a distância da região à margem inferior da janela.
Left	Inteiro	Define a distância da região à margem esquerda da janela.
Right	Inteiro	Define a distância da região à margem direita da janela.
background-color	Cor	Define a cor de fundo da região. Se este atributo não estiver definido será usado o valor default que é transparente.
Fit	"fill", "hidden", "scroll", "slice"	Define o comportamento no caso de a altura e a largura intrínsecas de um objecto multimédia visual serem diferentes dos valores especificados pelos atributos de altura e largura do elemento **region**. O valor default é "hidden".
z-index	Inteiro	Define a ordem pela qual cada região se sobrepõe.

Tabela 1 - Principais atributos dos elementos root-layout e region

2.1.2 Temporização dos Elementos Multimédia

A definição dos elementos multimédia que aparecem no documento SMIL é feita na secção **body**. No entanto, é nesta secção que se define também o *timing* destes mesmos elementos. Os *timings* dos elementos multimédia presentes nos documentos podem ser definidos através de atributos nos próprios elementos e através dos elementos **par** e **seq**. A temporização dos elementos multimédia assume um papel central em toda a especificação do SMIL, uma vez que esta é a sua principal característica.

O elemento sequencial (**seq**) indica que para todos os seus filhos será apenas considera uma linha temporal, ou seja, o seu filho seguinte apenas começará a ser reproduzido depois de o actual ter terminado. Desta forma, todos os elementos filho serão tocados em sequência, de acordo com os seus atributos. Este elemento possui alguns atributos que se encontram descritos na tabela 2.

Atributo	Tipo de Dados	Descrição
Begin	Tempo	Define especificamente o atraso com que este elemento será reproduzido.
End	Tempo	Define especificamente quando é que este elemento pára de ser reproduzido.
Dur	Tempo	Define especificamente a duração do elemento.
Id	Texto	Representa o identificador do elemento.
repeatCount	Inteiro ou "indefinite"	Define o número de vezes que a sequência será reproduzida. O valor default é 1.

Tabela 2 - Principais atributos do elemento seq

O elemento paralelo (**par**) indica que será criada uma nova linha temporal para os seus filhos, ou seja, cada um dos seus elementos filho será reproduzido em simultâneo. Este elemento pára de ser reproduzido assim que todos os seus filhos acabem de ser reproduzidos, excepto se for definida algum tipo de sincronização sobre nos seus atributos. A tabela 3 mostra alguns dos atributos possíveis para este elemento.

Atributo	Tipo de Dados	Descrição
Begin	Tempo	Define especificamente o atraso com que este elemento será reproduzido.
End	Tempo	Define especificamente quando é que este elemento pára de ser reproduzido.
Dur	Tempo	Define especificamente a duração do elemento.
Id	Texto	Representa o identificador do elemento.
repeatCount	Inteiro ou "indefinite"	Define o número de vezes que este elemento será reproduzido. O valor default é 1.
Endsync	id, "first" ou "last"	Define a sincronização entre os seus elementos filho. O valor default é "last".

Tabela 3 - Principais atributos do elemento par

Apesar de estes serem os principais elementos de temporização da linguagem estes não são obrigatórios nos documentos, sendo que, sempre que estes não se encontrem definidos nos documentos, será assumido por defeito que os elementos multimédia se encontram em sequência.

Para além dos elementos **par** e **seq** existe ainda o elemento exclusivo (**excl**) que tem por base o elemento **par**, no entanto apenas permite que um dos seus filhos seja reproduzido em qualquer momento. Se algum elemento filho está a ser reproduzido quando outro começa a ser reproduzido, o primeiro ou faz *pause* ou pára de ser reproduzido. Este elemento não é muito usado e fica

fora do contexto deste interpretador, motivo pelo qual não será focado neste documento.

Estes elementos de temporização permitem ainda que os seus filhos sejam também outros elementos de temporização, aumentando o grau de detalhe na definição da duração dos elementos multimédia dos documentos.

2.1.3 Elementos do Tipo Objecto Multimédia

Uma vez que não é o objectivo da linguagem fazer o *render* dos diversos formatos dos tipos de média, esta linguagem apenas define elementos genéricos de média, passando esta tarefa para os *players*. Desta forma, não faz sentido falar-se dos formatos de média suportados pela linguagem, mas sim dos tipos de média que esta reconhece e o tipo de propriedades que é possível associar a estes. Existem seis tipos de objectos multimédia nesta linguagem:

- Animação (**animation**) – Representa uma animação, como por exemplo um vector de gráficos animado;
- Áudio (**audio**) – Representa *clips* de áudio;
- Imagem (**img**) – Representa uma imagem fixa, do tipo PNG ou JPEG;
- Vídeo (**video**) - Representa *clips* de vídeo;
- Texto (**text**) – Representa uma referência a um texto externo;
- *Stream* de texto (**textstream**) – Representa um documento de texto externo com informações de temporização.

Para além destes seis tipos de objectos, existe ainda um tipo genérico que permite referir outros elementos multimédia que não sejam dos tipos anteriores. Este elemento é o **ref**.

Todos os objectos do tipo multimédia têm um atributo (**src**) no qual se define um URL para o local onde este objecto se encontra. Para além deste atributo, existe ainda um atributo opcional, o **type**, que permite indicar qual é o tipo de conteúdo do objecto referido no **src**. O atributo **type** aceita valores como “RTSP”, “HTTP” ou “FTP”. Os objectos multimédia possuem ainda outros atributos que se encontram descritos na tabela 4.

Atributos	Tipo de Dados	Descrição
Region	Id de uma região	Define em que região o objecto multimédia será reproduzido.
Begin	Tempo	Indica o tempo que deverá demorar este elemento para ser reproduzido depois de estar activo.
End	Tempo	Indica ao fim de quanto tempo este elemento deixa de estar activo
Dur	Tempo	Define a duração do elemento.
repeatDur	Tempo	Repete o elemento pelo tempo especificado.
repeatCount	Inteiro ou "indefinite"	Repete o elemento o número de vezes especificado.
Max	Tempo	Define o máximo de tempo que um objecto se encontra no estado activo.
Min	Tempo	Define o mínimo de tempo que um objecto se encontra no estado activo.
Fit	"fill", "hidden", "meet", "scroll" ou "slice"	Especifica o que acontece quando um elemento é maior ou menos que a área onde é reproduzido.

Tabela 4 - Principais atributos dos objectos multimédia

Tal como é possível ver pela tabela 4, também é possível definir *timings* nos próprios objectos multimédia, através de atributos como o **begin**, o **end**, o **dur**, **repeatDur**, **repeatCount**, **max** e **min**. O atributo **fit** apenas faz sentido quando se refere a imagens ou vídeos.

2.2 Exemplos de Documentos SMIL

Sendo o SMIL uma linguagem usada para controlar apresentações multimédia, de seguida apresenta-se alguns exemplos que demonstram de que forma é possível usar esta linguagem para implementar algumas das características no servidor de *streaming*, como o *picture-in-picture* e o *logo-insertion*.

A figura 2 mostra um exemplo de um documento SMIL que implementa a funcionalidade de *picture-in-picture*, usando duas regiões diferentes, uma mais pequena em cima da região principal. Ambas as regiões irão tocar o mesmo vídeo, mas a região de *preview*, a de menor dimensão, irá começar a tocar o vídeo dois segundos antes da região principal.

```
<?xml version="1.0">
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.1//EN"
           http://www.w3c.org/2005/SMIL21/SMIL21.dtd>
<smil>
    <head>
        <layout>
            <root-layout id="Geral" width="300" height="300"
                backgroundColor="white"/>
            <region id="preview" width="100" height="100"
                top="190" left="10" backgroundColor="black"
                z-index="1"/>
        </layout>
    </head>
    <body>
        <par>
            <video src="data/video.mpg" region="Geral" begin="2s"
                fit="fill"/>
            <video src="data/video.mpg" region="preview" fit="fill"/>
        </par>
    </body>
</smil>
```

Figura 2 - Exemplo de um documento SMIL que implementa a funcionalidade de *picture-in-picture*

Tal como podemos ver neste exemplo, dentro do elemento **head** temos a definição do *layout* do documento. Nesta definição são definidas duas regiões, a região "Geral" e a região "Preview". A primeira região a ser definida tem que ser o **root-layout**, uma vez que é esta região que irá definir o tamanho da janela. Usando os atributos **width** e **height** são definidos os tamanhos das regiões. Adicionalmente, a definição da região "Preview" usa também os atributos **top** e **left** para definir a sua posição na janela e o atributo **z-index** para especificar que fica no topo, ou seja, em cima das restantes regiões definidas.

Na secção **body**, o elemento **par** é usado pois ambos os vídeos serão reproduzidos em simultâneo. Assim, dentro do elemento **par** são definidos os dois vídeos. O primeiro vídeo definido é aquele que será reproduzido na região "Geral" e, como este vídeo terá um atraso de dois segundos em relação ao vídeo anterior, é usado o atributo **begin** para definir este atraso. Em ambos os vídeos é usado o atributo **fit** definido como "fill" para indicar que os vídeo deverão ser redimensionados de forma a que ocupem toda a área da região onde serão reproduzidos. As figuras 3 e 4 apresentam o resultado obtido com este documento.

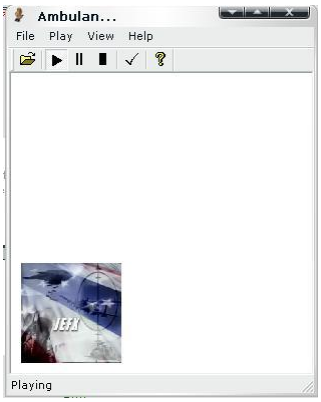

Figura 3 - *Picture-in-Picture* com *preview* na região mais pequena

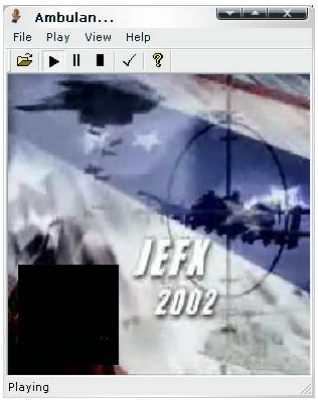

Figura 4 - *Picture-in-Picture* passados três segundos do inícioda reprodução

A funcionalidade de *logo-insertion* também pode ser conseguida de forma bastante semelhante, conforme é possível ver na figura 5.

```
<?xml version="1.0">
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.1//EN"
           http://www.w3c.org/2005/SMIL21/SMIL21.dtd>
<smil>
    <head>
        <layout>
            <root-layout id="Geral" width="300" height="300"
                backgroundColor="white"/>
            <region id="logo-area" width="50" height="50"
                top="10" left="10" z-index="1"/>
        </layout>
    </head>
    <body>
        <par>
            <img src="data/logo.png" region="logo-area"
                dur="indefinite"/>
            <video src="data/video.mpg" region="Geral" fit="fill"/>
        </par>
    </body>
</smil>
```

Figura 5 – Exemplo de um documento SMIL que implementa *logo-insertion*

Tal como no exemplo da figura 2, neste exemplo são usadas duas regiões, uma principal com o nome “Geral” e outra secundária com o nome “logo-area”. À semelhança do exemplo anterior, a definição das regiões é igual excepto a “logo-area” cuja posição na janela é diferente. No **body** do documento temos a definição do logo que queremos inserir. Isto é feito através do elemento **img**. Como é necessário que a imagem seja apresentada durante toda a apresentação do vídeo, a duração desta é definida como “indefinite”. A duração da apresentação da imagem tem que ser obrigatoriamente definida, caso contrário será usado a duração por defeito que é de zero segundos. Adicionalmente poderia ter-se definido o atributo **endsync** no elemento **par** como “first”, indicando que assim que o primeiro elemento filho (o vídeo ou a imagem) terminasse, os restantes filhos também terminariam.

O resultado da apresentação do documento SMIL da figura 5 pode ser visto na figura 6.

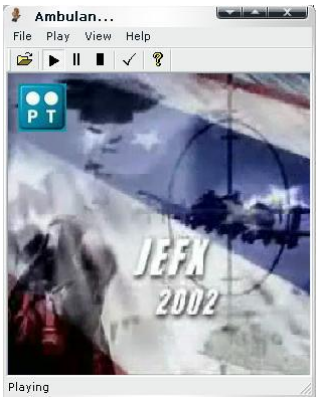

Figura 6 - *Logo-insertion* conforme descrito na figura 5

3. Análise Tecnológica dos *Players* de SMIL

Após o estudo da linguagem SMIL e sua especificação, tornou-se necessária uma certa experimentação da linguagem a fim de consolidar os conhecimentos sobre esta. Para tal foram estudados alguns *players* de SMIL e suas capacidades, para essa experimentação e, numa fase posterior, para ver de que forma estes se encontravam organizados a fim de servirem como exemplo e/ou guia para a criação do interpretador.

Foram estudados seis *players* de SMIL, entre os quais três dedicam-se exclusivamente a reproduzir este tipo de documentos, ao passo que os restantes suportam outros tipos de formatos multimédia e, por vezes, precisam da instalação de um *plug-in* para suportarem este tipo de documentos. Os *players* e ferramentas para SMIL estudados foram:

- Ambulant Player;
- Helix Player[5];
- Schmunzel[6];
- X-Smiles[7];
- GRiNS[8];
- QuickTime[9].

De seguida é apresentada uma descrição de cada um destes *players* e quais as suas funcionalidades. Depois segue-se uma conclusão onde é feita uma comparação dos principais pontos fortes e fracos de cada ferramenta analisada e é justificada a escolha do Ambulant Player como guia para o interpretador.

3.1 Características Procuradas

O *player* procurado para servir de guia para a construção do processador de SMIL deverá cumprir os seguintes requisitos:

1. Ser open-source;
2. Ser independente do sistema operativo;
3. Ser implementado em C++ ou Java;

4. Implementar a recomendação SMIL 2.1 ou superior;
5. Estar bem documentado;
6. Ter uma equipa de suporte activa e capaz de ajudar da resolução de problemas de instalação e implementação ou tirar dúvidas sobre a arquitectura.

Os formatos de media suportados, em especial para o áudio, vídeo e imagens, por cada um dos *players* não é relevante para este estudo, uma vez que estes *players* recorrem frequentemente a bibliotecas externas para fazerem o *render* destes, podendo os *players* em muitos casos ser estendidos de forma a suportarem outros formatos através de outras bibliotecas.

O *player* procurado deverá ser open-source, uma vez que se pretende que este sirva de guia para o processador e se possível reutilizar algum código. Como se pretende que o processador seja implementado tanto para Linux como para Windows, é preferencial que o *player* seja independente do sistema operativo, de forma a facilitar o cumprimento de este objectivo. A linguagem de programação na qual o *player* foi implementado não é uma questão essencial nesta pesquisa, no entanto uma linguagem orientada a objectos é preferível para a reutilização de código, pois facilita esta tarefa. Pretende-se que o *player* implemente uma recomendação SMIL igual ou superior à 2.1 pois este é também um requisito do interpretador.

O facto de o player se encontrar bem documentado e de ter uma boa equipa de suporte são requisitos fundamentais nesta escolha, pois a análise do código fonte por si só não é elucidativa do modo de funcionamento do *player* e a documentação e a equipa de suporte auxiliam neste passo, reduzindo o tempo de desenvolvimento.

3.2 Ambulant Player

O Ambulant Player é uma ferramenta de reprodução exclusiva de documentos SMIL. Na sua versão base, esta ferramenta reproduz apenas este tipo de documentos, no entanto, esta pode ser compilada como *plug-in* para o browser Mozilla Firefox (Windows e Linux) e para o browser Safari (MAC OS X). O Ambulant Player vai actualmente na sua versão 2.0.1 e é neste momento

o único *player* que lê documentos SMIL 3.0, ou seja, que aceita e segue a recomendação do W3C para o SMIL 3.0, que foi aceite a 1 de Dezembro de 2008. Este é um *player* a que se pode chamar de puro, uma vez que mantém a especificação original do SMIL e todos os seus módulos, em vez de estender a linguagem para uma linguagem sua, a fim de colmatar algumas lacunas na sua implementação.

O Ambulant Player é uma ferramenta open-source e independente de sistema operativo, ou seja, pode ser instalado em qualquer sistema operativo (Windows, Linux, Mac OS X, entre outros) e o seu código fonte encontra-se disponível segundo uma LGPL (GNU Lesser General Public License). Este *player* foi desenvolvido para programadores que procuram um *player* open-source sobre o qual possam desenvolver sistemas de alto nível para edição ou integração de conteúdos ou sobre o qual possam criar ou estender novos componentes de *networking* e transporte de multimédia.

Este player foi escrito em C++ e contém uma API que permite a sua integração em Python. No que diz respeito à sua organização, a arquitectura deste *player* pode ser dividida em duas camadas, uma de alto nível que é independente do sistema operativo e outra de baixo nível dependente do sistema operativo. Ao contrário de outros *players* que apenas são usados para reproduzir música e vídeos, este *player* é usado em escolas para a visualização de documentos SMIL simples e na aprendizagem de SMIL para o seu uso em MMS (Multimédia Messaging Service). A biblioteca do Ambulant Player foi usada como *renderer* no projecto AMIS que tinha por objectivo criar livros "falantes" e também para a visualização de informação num projecto que pretendia facilitar a cooperação entre veterinários.

3.3 Helix Player

O Helix Player é um projecto que pretende desenvolver software que seja capaz de reproduzir média em vários formatos e ajudar na sua produção e distribuição. O Helix DNA Client impulsionou algumas aplicações como o RealPlayer, o RealPlayer Mobile e o HelixPlayer, que são utilizados em larga

escala. Este *player* vai na sua versão 11.0 e suporta documentos SMIL até à recomendação 2.0.

Este player é open-source e foi escrito em C++ mas, no entanto, não é independente do sistema operativo, correndo apenas em sistemas UNIX. Em termos de documentação, todo o sistema encontra-se bem documentado, estando esta disponível na página oficial do *player*.

3.4 Schmunzel

O Schmunzel é um *player* de SMIL totalmente escrito em Java, logo é independente do sistema operativo. Não existem informações sobre se este é open-source ou não e a sua página oficial não parece ser actualizada à muito tempo. Este *player* foi desenvolvido pela SunTREC Salzburgo e apenas suporta parte da recomendação do SMIL 1.0, no entanto acrescenta a esta algumas funcionalidades, estendendo a especificação do SMIL. O desenvolvimento e actualização deste *player* parecem ter sido descontinuadas.

3.5 X-Smiles

O X-Smiles antes de ser um *player* de SMIL é um *browser* que é capaz de lidar com documentos XML. Ele faz o *parsing* e transformações a todos os documentos XML bem formados. Desta forma, este documento não só é capaz de reproduzir documentos SMIL até à recomendação 2.0 como também consegue reproduzir documentos SVG, entre outros formatos.

Este *player* vai neste momento na sua versão 1.2 (lançada a 14 de Março de 2008), é open-source e independente do sistema operativo, uma vez que foi escrito em Java. Este *player* encontra-se bem documentado e não faltam informações sobre a arquitectura deste na sua documentação.

3.6 GRiNS

O GRiNS é um *player* que vai na versão 2.2 e cobre a recomendação SMIL 2.0. Este *player* não é open-source nem independente do sistema operativo, encontrando-se apenas disponível para Windows. Apesar de comercial, esta ferramenta é muito referenciada quando se fala de SMIL, uma vez que disponibiliza um editor gráfico para a criação de documentos SMIL e vídeos para o RealOne.

Sendo esta ferramenta comercial e devido à falta de informações sobre esta, este *player* fica à partida excluído desta pesquisa uma vez que não cumpre um dos seus requisitos essenciais.

3.7 QuickTime

O QuickTime é um *player* que fornece uma API rica para ajudar a reproduzir, importar, exportar, modificar e capturar muitos tipos de media. A partir da sua versão 4.1 suporta SMIL e permite que estes documentos sejam reproduzidos como se fossem filmes do QuickTime. Contudo, não se encontra disponível qual a especificação de SMIL suportada pelo *player*, para além de este não ser nem open-source nem independente do sistema operativo (apenas se encontra disponível para Windows e Mac OS X).

3.8 Player de Referencia Escolhido

Após a análise cuidada de cada um dos *players*, o que se destaca mais de todos estes foi o Ambulant Player. A tabela 5 pretende resumir as informações adquiridas sobre cada *player* e fornecer uma visão geral dos pontos fortes e fracos de cada um.

	Ambulant Player	Helix Player	Schmunzel	X-Smiles	GRiNS	QuickTime
Versão Estudada	2.0.1 (Nov. 08)	11.0 (Abr. 08)	0.4 (Dez. 00)	1.2 (Mar. 08)	2.2 (Mai 02)	7
SMIL 1.0	Sim	Sim	Sim	Sim	Sim	Sem Informação
SMIL 2.0	Sim	Sim	Não	Sim	Sim	Sem Informação
SMIL 2.1	Sim	Não	Não	Não	Não	Sem Informação
SMIL 3.0	Sim	Não	Não	Não	Não	Sem Informação
Independente do S.O.	Sim	Não	Sim	Sim	Não	Não
Open-Source	Sim (LGPL)	Sim	Sem Informação	Sim (Apache)	Não ($95)	Não ($0)
Linguagem usada	C++	C/C++	Java	Java	Sem Informa-ção	Java/C

Tabela 5 - Características dos players estudados

Tal como se pode ver pela tabela 5, o Ambulant Player e o X-Smiles cumprem os dois primeiros requisitos, ou seja, são open-source e independentes do sistema operativo. Desta forma, é possível descartar-se o Helix Player, pois apenas se encontra disponível para Linux, o GRiNS, pois apenas se encontra disponível para Windows é não é open-source, e o QuickTime, pelos mesmos motivos que o GRiNS. O Schmunzel apenas suporta SMIL 1.0 e foi descontinuado, logo não cumpre os requisitos 4 e 6, motivo pelo qual também não representa uma opção viável para guia do processador de SMIL.

Deste modo apenas restam duas opções: o Ambulant Player e o X-Smiles. Uma vez que o Ambulant Player suporta todas as especificações de SMIL existentes, incluindo a 3.0, tem uma excelente equipa de desenvolvimento sempre disponível para esclarecer dúvidas e encontra-se bem documentado, este é a melhor opção.

4. Arquitectura do Ambulant Player

Tal como foi dito na secção 3.8, o Ambulant Player foi a escolha ideal para guia do interpretador que se pretende implementar. Após a escolha do *player* guia para o interpretador, o próximo passo é estudar a sua arquitectura para que seja possível usar o maior número de componentes deste. A documentação do Ambulant Player foi profundamente estudada e várias provas de conceito foram feitas até que fosse possível encontrar a sua arquitectura.

4.1 Princípios de Desenvolvimento

Após a leitura aprofundada da documentação deste *player* foi possível conhecer algumas das decisões tomadas pela equipa que o desenvolveu e ter uma primeira noção da arquitectura geral do Ambulant Player.

De forma a poder reproduzir documentos SMIL, este *player* necessita de um *global playback engine* que contém todos os restantes componentes associado a ele. Para tal neste *player* foi criado o objecto **player**, que é responsável por controlar todos os aspectos do *playback* de um único documento SMIL. Adicionalmente, este *player* usa *factories* para criar *file readers, renderes* e janelas. Estas *factories* são povoadas pelo programa principal e usadas durante a reprodução do documento.

Uma vez que o Ambulant Player pretende ser um sistema vocacionado para a investigação, todos os seus componentes têm que ser facilmente substituídos sem interferirem com o resto do sistema, para que seja possível aos investigadores concentrarem-se num só assunto, mantendo o resto do sistema sem problemas. Desta forma, existe uma API bem definida entre as diversas partes do sistema e um conjunto funções padrão para criar os diversos objectos.

A arquitectura do *player* é orientada a eventos, com um pequeno número de *working threads* que recolhem os eventos de uma *event queue*. Em alternativa, e segundo a equipa de desenvolvimento do Ambulant Player, poderiam ter sido usados diversos *working threads*, De acordo com os

mesmos, é mais fácil para um objecto que pretende usar múltiplos *threads* fazê-lo numa infra-estrutura do tipo de eventos ao invés de uma *multi-threading*. Uma arquitectura deste tipo exige um esquema de prioridades elaborado e expressivo o suficiente para que haja a melhor ordem de execução dos eventos.

De uma forma geral, a maneira usada pelo *player* para lidar com código dependente do sistema operativo é criando uma classe abstracta independente do sistema operativo e depois as respectivas subclasses dependentes do sistema operativo. Existe uma função da *factory* que cria uma instância dependente do sistema operativo e retorna-a convertida na sua classe base independente do sistema operativo. Esta característica do *player* permite que se afirme que este pode ser dividido em duas camadas, uma primeira de alto nível independente do sistema operativo e uma segunda de um nível inferior dependente do sistema operativo.

4.2 Ferramentas de Terceiros

O Ambulant Player utiliza algumas ferramentas de terceiros, entre as quais o Expat[11] e o Xerces[12] para processamento dos documentos XML, o FFmpeg[13] e o SDL[14] para processar *streams* de áudio e vídeo, o Live555[15] para suportar *streaming* em RTP/RTCP, RTSP e SIP, o GStreamer[16] para controlar elementos de media e o GTK[17] para a criação do ambiente gráfico para o utilizador. De seguida é descrito em que consiste cada uma destas bibliotecas e de que forma são usadas no Ambulant Player.

4.2.1 Expat

O Expat é uma biblioteca escrita em C que faz o *parsing* de documentos XML. Esta biblioteca é o *parser* subjacente a alguns projectos *open-source* como o projecto Mozilla ou o *parser* XML::Parser em pearl. Esta biblioteca é muito rápida a fazer o *parsing* de documentos XML e é conhecida por ter elevados padrões de fiabilidade, robustez e exactidão.

O Expat é uma biblioteca de parsing orientada a *streams*, ou seja, o documento XML não necessita de ser completamente carregado para memória para ser processado. A utilização deste *parser* assenta, numa primeira fase, no registo de *handlers* para os diversos casos possíveis no documento e, de seguida, o documento começa a ser alimentado ao *parser*. Sempre que o *parser* reconhece parte do documento chama o respectivo *handler* (se tiver algum registado). Desta forma o *parsing* do documento é feito por partes, de forma eficiente, e permitindo assim fazer o *parsing* de documentos de grandes dimensões, cujo tamanho não permite que sejam carregados para a memória.

4.2.2 Xerces

O Xerces é uma colecção de bibliotecas que manipulam documentos XML, fazendo o *parsing* e a validação destes. As suas implementações estão disponíveis para Java, C++ e Perl. O Ambulant Player usa a versão escrita em C++, que está sob uma *Apache Software License.*

O Xerces permite usar as convenções DOM, SAX e SAX2 e faz a validação dos documentos XML usando os Schemas da respectiva linguagem. O *parsing* do documento XML pode ser feito de forma progressiva, no entanto este é carregado na sua totalidade para a memória antes de se iniciar o *parsing*. Tal como no Expat, o processamento dos documentos também tem por base o registo de *handlers* que são invocados quando o *parser* reconhece alguma parte do documento.

O Ambulant Player permite escolher se se pretende usar o Expat ou o Xerces, estando o Expat associado a melhor desempenho no *parsing* do documento e recomendado para documentos SMIL de maiores dimensões, por oposição ao Xerces que se encontra associado a uma validação mais forte do documento e não recomendado para documentos SMIL de elevadas dimensões.

4.2.3 Ffmpeg

O ffmpeg é uma biblioteca que permite converter ficheiros de som e de vídeo de forma muito rápida. Esta biblioteca permite também capturar áudio e vídeo em directo. O ffmpeg permite ainda criar *streams* de áudio e vídeo a partir de diversos formatos.

No Ambulant Player, esta biblioteca é usada para o *render* de áudio e video. O *player* permite que o utilizador escolha entre o ffmpeg e o SDL para reproduzir o áudio e o vídeo, sendo o ffmpeg a opção escolhida por defeito.

4.2.4 SDL

A Simple DirectMedia Layer (SDL) é uma biblioteca escrita em C multiplataforma que permite uma abstracção em várias plataformas de gráficos e sons, entre outras funcionalidades. Esta biblioteca encontra-se disponível sob uma GNU LGPL e é usada recorrentemente em jogos e emuladores.

No Ambulant Player esta biblioteca apresenta-se como solução de recurso para o uso do ffmpeg para a reprodução de elementos de áudio e video.

4.2.5 Live555

O Live555 é uma biblioteca escrita em C++ que permite manipular protocolos como RTP/RTCP, RTSP, ou SIP para *streaming*, permitindo ainda manipular formatos de áudio e vídeo como MPEG ou H.263+. Esta biblioteca tanto permite implementar lado do cliente de *streamings* de media como o lado do servidor. Um exemplo da aplicação desta biblioteca é o *Live555 Media Server*, que é um servidor RTSP que permite fazer o *streaming* de vários tipos de ficheiros multimédia.

No Ambulant Player esta biblioteca é usada para permitir o uso de *streams* de áudio e de video sendo opcional o seu uso.

4.2.6 GStreamer

O GStreamer é uma *framework* para o desenvolvimento de aplicações que lidam com *streams* multimédia, como áudio ou vídeo. Este foi escrito em C e é multiplataforma, estando disponível para Linux, Solaris, MAC OS X, Microsoft Windows e OS/400. A *framework* encontra-se disponível sob uma GNU GPL e foi usada em aplicações como o Totem, Rhythmbox, Kaffeine ou Exaile.

No Ambulant Player esta *framework* apresenta uma alternativa ao uso do ffmpeg.

4.2.7 GTK

O GTK é um conjunto de ferramentas multiplataforma para a criação de interfaces gráficos. Estas ferramentas foram escritas na linguagem C mas, no entanto, permitem a sua integração em outras linguagens como o C++, Python ou C#. As ferramentas que compõem o GTK encontram-se licenciadas sob uma GNU LGPL.

No Ambulant Player o GTK é usado na criação do ambiente gráfico no qual o utilizador pode visualizar os documentos SMIL.

4.3 Arquitectura e Design

A arquitectura do Ambulant Player pode ser dividida em duas partes, a interface gráfica para o utilizador (GUI) e a sua biblioteca de sistema (libAmbulant). A libAmbulant é responsável pelo processamento dos documentos SMIL e sua reprodução ao passo que o GUI opera sobre esta biblioteca e cria a interface com o utilizador. Uma vez que o interpretador dispensa o uso de interface gráfica, a análise da parte referente ao GUI não foi aprofundada, limitando-se à percepção de como operar com a libAmbulant.

De forma a poder reproduzir um documento SMIL, a libAmbulant necessita de uma instância do seu *player,* para controlar o *playback* do documento, de várias *factories*, responsáveis por criar *renderes, file readers* e *parsers* entre outros, e de um *embedder*, responsável por controlar os eventos gerados durante a reprodução do documento. A figura 7 apresenta a arquitectura simplificada de uma aplicação que usa a libAmbulant para ler e reproduzir um documento SMIL.

Um programa simples que utilize esta biblioteca precisa de, em primeiro lugar, inicializar as cinco *factories* básicas necessárias. Estas cinco *factories* são genéricas e, dependendo do sistema operativo usado, instanciam a *factory* especifica que será usada para esse fim. A *parser factory* é usada para realizar o *parsing* do documento SMIL e gerar a árvore de DOM, usando a *node factory*. Tal como foi dito na secção 2 deste capítulo, o Ambulant Player usa dois *parsers* de terceiros, o Expat e o Xerces.

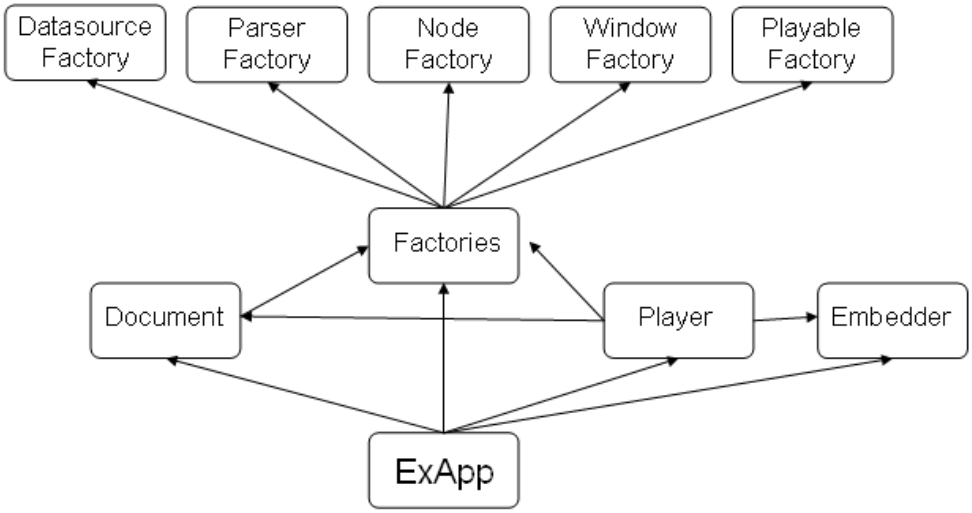

Figura 7 - Arquitectura simplificada de uma aplicação que use a libAmbulant

Desta forma, estão disponíveis duas *factories* para povoar a parser factory, uma disponibilizando o Expat para o *parsing* do documento e a outra disponibilizando o Xerces. O Expat faz um *parsing* mais rápido do documento SMIL ao passo que o Xerces faz uma validação mais profunda e forte do documento. A opção por defeito da biblioteca é o uso do Expat como *parser*. A Window Factory é usada pelo *player* sempre que precisa de criar uma nova janela para reproduzir um elemento do documento SMIL. A Playable Factory é

usada pelo *player* para reproduzir os nós do documento SMIL, ou seja, para criar objectos do tipo playable e controlar as suas acções quando reproduzidos pelo *player*. A Datasource Factory é usada para guardar os diferentes *renderes* para os diferentes tipos de media.

Depois da inicialização destas *factories*, o passo seguinte é carregar o documento SMIL. A classe Document contém três métodos para carregar documentos:

1. Através do nome do ficheiro que contém o documento SMIL;
2. Através de um URL que aponte para o documento SMIL;
3. Através de uma *string* que com o referido documento.

Cada um destes três métodos recebe ainda como parâmetro um apontador para a instância Factories. Depois de carregado o documento, é possível criar uma instância do player passando um apontador para a instância da classe Factories, um documento e um apontador para a classe do Embedder. O Embedder é uma classe que contém *callbacks* que são usados quando o *player* inicia ou pára a reprodução de um documento, por exemplo. Depois de criada a instância do *player* é necessário inicializá-lo. Esta inicialização basicamente inicia o *scheduler* e o processador de eventos usados pelo *player* para controlar a apresentação dos elementos multimédia.

Após a inicialização do *player* não é necessário realizar mais nenhum passo para se iniciar a reprodução do documento SMIL, bastando apenas invocar o método start deste.

5. Desenvolvimento do Interpretador

Carecendo de alguma documentação, uma análise concreta do Ambulant Player foi necessária. Como não havia informações sobre o funcionamento de várias instâncias do *player* em diferentes *threads*, um dos primeiros passos do desenvolvimento do interpretador foi criar uma prova de conceito que permitisse avaliar esta possibilidade.

Após a criação desta prova de conceito foram estudados os requisitos e foi feita a modelação, usando a metodologia ICONIX, do interpretador, passando-se de seguida para o desenvolvimento da solução.

5.1 Requisitos

O interpretador de SMIL implementado terá que cumprir alguns requisitos os quais serão apresentados nesta secção. O CSMIL tinha vários requisitos, desde requisitos operacionais, requisitos de desempenho, requisitos funcionais… Alguns requisitos advêm dos requisitos do WSS e outros são específicos para o interpretador.

No que diz respeito ao interpretador, este tem os seguintes requisitos:

1. Suporte para documentos SMIL que sigam a norma 2.1 ou superior: o CSMIL deverá realizar o *parsing* de documentos SMIL que sigam a norma 2.1 ou superior.
2. Descrição de sessões a pedido do session manager: o CSMIL deverá realizar o *parsing* de um documento SMIL a pedido do session manager e fornecer a este informações referentes à duração da sessão, uso de *picture-in-picture* e de endereços RTSP.
3. Leitura de configurações a partir de ficheiro: o interpretador deverá carregar alguns parâmetros a partir de um ficheiro de configuração. Entre estes parâmetros encontra-se a localização do ficheiro de *logging*, o nível de *log*, o intervalo de recepção e envio de *heartbeats* e o porto de comunicação com o módulo de gestão de sessões.
4. Registo de informações das sessões: o CSMIL deverá implementar um sistema de *logging* para registo do estado de cada sessão e do

seu funcionamento interno, facilitando assim a correcção de erros e análise dos acontecimentos em caso de falha.

5. Controlo de estado por *heartbeat*: o interpretador deverá implementar um sistema tipo *heartbeat* Linux, no qual envia o seu estado ao session manager e indica que se encontra "vivo", permitindo assim fazer um controlo da disponibilidade deste.
6. Dados estatísticos: o CSMIL deverá disponibilizar o número total de sessões servidas, o número total de sessões que pode aceitar, o número de sessões simultâneas no momento, o número máximo de sessões simultâneas e qual esse momento e o número de erros ocorridos nas sessões até ao momento sempre que solicitado pelo session manager.
7. Controlo da reprodução dos elementos multimédia da sessão: o CSMIL deverá controlar o início da reprodução de elementos multimédia de acordo com o documento SMIL carregado para essa sessão.
8. Controlo da paragem de reprodução dos elementos multimédia da sessão: o CSMIL deverá controlar o término da reprodução dos elementos multimédia de acordo com o especificado no documento SMIL que descreve a sessão.
9. Carga: o CSMIL deverá ser capaz de suportar a criação de cinquenta novas sessões por segundo e de gerir quinhentas sessões em simultâneo;
10. *Picture-in-Picture*: o CSMIL deverá suportar a funcionalidade de *picture-in-pictrure,* ou seja, permitir que dois vídeos sejam reproduzidos em simultâneo, um como fundo e outro numa pequena janela em primeiro plano.
11. *Text-Overlay:* o interpretador deverá suportar *text-overlay*, ou seja, deverá ter a capacidade de inserir elementos de texto sobre elementos de vídeo.
12. *Logo-insertion:* o interpretador deverá ter a capacidade de inserir logótipos sobre vídeos, de acordo com a sua especificação no documento SMIL referente à sessão.

13. *Pre/Pos-role:* o sistema deverá ter a capacidade de colocar anúncios publicitários antes e após a reprodução de vídeos.
14. Reprodução repetida de elementos multimédia: o interpretador SMIL deverá ter capacidade de suportar a repetição de elementos multimédia indefinidamente, um determinados número de vezes ou durante um certo intervalo de tempo.

5.2 Prova de Conceito

Tal como dito anteriormente, não existem documentadas tentativas prévias de colocar a libAmbulant num ambiente *multi-threading* de suporte a várias sessões de leitura de documentos SMIL. Esta prova de conceito pretendia ser uma pequena aplicação que implementava várias instâncias de *players* SMIL e que em vez de reproduzir os elementos multimédia mostrava uma mensagem a dizer o início e o término da sua reprodução. Pretendia-se também com esta prova de conceito adquirir conhecimentos sobre a compilação da libAmbulant e sua integração noutras aplicações, bem como perceber de que modos se interligam as diversas entidades necessárias à reprodução de um documento na biblioteca.

5.2.1 Objectivos

Tal como foi introduzido anteriormente, de uma forma geral esta prova de conceito pretendia verificar a possibilidade de colocar diferentes instâncias de *players* de SMIL a correr em simultâneo e perceber de que forma se conjugam os elementos da biblioteca para permitir a reprodução de documentos. Deste modo, a aplicação criada para esta prova de conceito deveria:

1. Compilar uma versão da libAmbulant para uma pequena aplicação, na qual se excluísse os elementos de rendering, desnecessários para o interpretador;

2. Implementar uma aplicação que crie várias instâncias de *players* SMIL usando a biblioteca do Ambulant Player, sem recurso à interface gráfica do *player*;
3. Cada instância do *player* deve escrever uma mensagem no ecrã com as informações referentes ao elemento que se encontra a reproduzir, em vez de reproduzir o elemento;
4. O uso de *factories* deverá ser restrito apenas aquelas que são necessárias para os requisitos da prova de conceito;

Com esta prova de conceito pretendia-se provar que:

1. É possível integrar a libAmbulant noutras aplicações;
2. É possível criar várias instâncias de *players* SMIL e executá-las em simultâneo sem que haja conflitos, ou seja, que não existem recursos partilhados entre as diferentes instâncias criadas ou que estes, caso existam, não entram em conflito por estarem partilhados;
3. É possível correr um *player* SMIL sem reproduzir elementos multimédia;
4. É possível compilar a libAmbulant com apenas algumas bibliotecas necessárias para o pretendido;
5. É possível aceder à informação referente aos recursos multimédia que se encontram a ser reproduzidos quando começa a reprodução destes;

Deste modo, esta prova de conceito apresenta um papel fulcral para demonstrar de que forma o Ambulant Player e a sua biblioteca podem ser usados noutras aplicações para além da reprodução de elementos multimédia descritos em documentos SMIL. Não existem documentadas outras utilizações deste *player*, não havendo também informações sobre de que forma integrar este noutros contextos, atribuindo um papel fulcral a esta prova de conceito.

Não sendo o principal objectivo desta, a prova de conceito também permitiu inferir sobre as capacidades de robustez e de carga da biblioteca. Desta forma é possível antever se seriam necessários alguns melhoramentos na biblioteca de forma a que os objectivos de carga e robustez do interpretador fossem cumpridos.

5.2.2 Arquitectura e Desenvolvimento

Tendo por base os objectivos que se pretendem cumprir com esta prova de conceito, uma estrutura simples foi pensada para o seu desenvolvimento. Uma *thread* principal tem como função gerir os novos *players* instanciados. Cada *player* corre num *thread* separado que numa primeira fase faz o *parsing* do documento, de seguida instancia um *player* e de seguida invoca o *start* deste dando inícioà reprodução. Sempre que uma instância do *player* termina, o método *done* da instância do *player* é invocado, sendo invocados os destrutores do *player* e terminado o *thread*.

A figura 8 representa a lógica de funcionamento pretendida com esta prova de conceito:

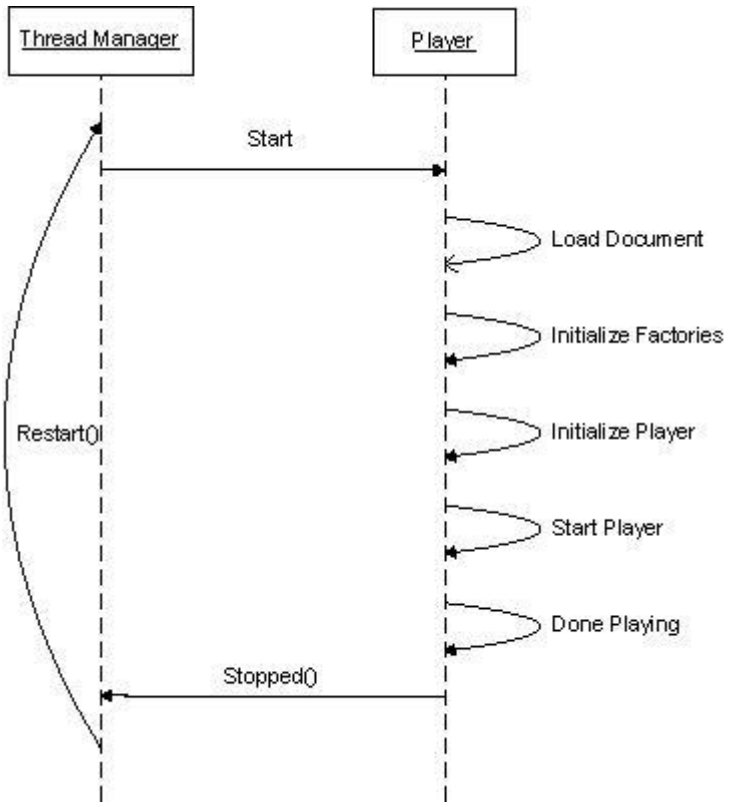

Figura 8 - Lógica de funcionamento da prova de conceito

Uma vez que as capacidades funcionais da biblioteca não estavam em teste, para facilitar a validação da prova de conceito o mesmo documento SMIL era carregado para todas as instâncias do *player*. O documento usado nesta prova de conceito reproduziria um documento SMIL com uma sequência de

dois vídeos, cada um com a duração de quinze segundos. O documento SMIL carregado encontra-se descrito na figura 9.

```
<?xml version=”1.0”>
<!DOCTYPE smil PUBLIC “-//W3C//DTD SMIL 2.1//EN”
           http://www.w3c.org/2005/SMIL21/SMIL21.dtd>
<smil>
    <head>
        <layout>
            <root-layout id=”Geral” width=”300” height=”300”/>
        </layout>
    </head>
    <body>
        <seq>
            <video src=”data/video1.mpg” region=”Geral” dur=”15s”/>
            <video src=”data/video2.mpg” region=”Geral” dur=”15s”/>
        </seq>
    </body>
</smil>
```

Figura 9 - Documento SMIL usado na prova de conceito

Assim sendo, assumindo que apenas seria reproduzido um documento, como resultado da prova de conceito deveria-se obter no ecrã uma mensagem no inícioda reprodução indicando que um elemento de vídeo seria reproduzido, que se encontrava na localização “data/” e que tinha o nome de “video1.mpg”, passado quinze segundo que esse mesmo vídeo terminava a sua reprodução e que se daria inícioà reprodução do segundo vídeo, e trinta segundos depois que este último vídeo terminava a sua reprodução, estando terminada a reprodução do documento.

5.2.3 Avaliação e Testes

Os testes realizados a este protótipo foram desenvolvido em dois ambientes diferentes, um ambiente Windows e um ambiente Linux. Desta forma pretendia-se também testar o funcionamento da biblioteca em diferentes sistemas operativos. A tabela 6 resume as características de ambos os sistemas.

	Windows	Linux
Versão do Sistema Operativo	Windows XP Professional, Service Pack 3	CentOS 5
Processador	Intel Pentium IV 2,66 GHz	Intel Xeon 5160 3,00 GHz
Memória RAM	1,00 GB	256 MB

Tabela 6 - Características dos ambientes de teste da prova de conceito

Tal como podemos ver pela tabela 6, apesar de o ambiente de testes ter mais memória RAM, o ambiente Linux tem um processador mais rápido o que pode levar a uma discrepância dos resultados. No entanto o ambiente Linux encontra-se numa máquina virtual pelo que os resultados obtidos neste ambiente podem ser penalizados.

O primeiro teste realizado consistia na reprodução do documento apenas uma vez de forma a que se pudesse testar o correcto funcionamento da prova de conceito. Este teste foi realizado com sucesso em ambos os ambientes. Um segundo teste foi pensado de forma a testar a partilha de dados entre instâncias diferentes dos *players* a correr em *threads* diferentes. Deste modo, este teste instanciava duas sessões em *threads* diferentes e corria-os em simultâneo. Apesar de as sessões terem corrido sem problemas e os elementos multimédia terem sido reproduzidos com a temporização correcta, foi detectado que os identificadores dos elementos multimédia eram diferentes apesar de o *parsing* e da reprodução dos elementos ter sido feita em simultâneo. Depois de alguma investigação conclui que o identificador de cada nó da árvore DOM era partilhado por todas as instâncias criadas da libAmbulant, sendo este valor incrementado depois de atribuído. No entanto mais testes foram realizados e sempre com atenção a este problema para se perceber de que forma este podia influenciar a aplicação pretendida para a biblioteca.

Um terceiro teste foi implementado para que a robustez da biblioteca pudesse ser testada, ou seja, se é possível correr vários *players* em simultâneo sem fugas de memória ou sem que ocorram erros. O primeiro teste deste tipo colocava duas sessões em simultâneo com a duração de trinta segundos, esperava cinco segundos, colocava mais duas sessões e assim sucessivamente. O segundo teste em vez de duas sessões colocava cinco

sessões em simultâneo. O terceiro teste colocava dez sessões em simultâneo. A tabela 7 resume os parâmetros dos três testes.

Teste	Sessões Simultâneas	Intervalo entre séries
1	2	5 segundos
2	5	5 segundos
3	10	5 segundos

Tabela 7 - Resumo dos parâmetros dos testes de robustez efectuados à prova de conceito

Os resultados obtidos com este conjunto de testes variam de acordo com o ambiente de teste e não foram os ideias. Esperava-se que fossem reproduzidas cerca de um milhão de sessões, ou seja, que a prova de conceito fosse parada por influência do utilizador e não devido à ocorrência de erros. Os primeiros testes foram realizados em ambiente Windows e os resultados obtidos encontram-se resumidos na tabela 8 e quais os motivos te paragem.

Número do Ensaio	Teste 1		Teste 2		Teste 3	
	Valor	Caso de Paragem	Valor	Caso de Paragem	Valor	Caso de Paragem
1	552	Erro a criar *threads*	474	Erro a criar *threads*	467	Erro a criar *threads*
2	968	Segmentation Fault	935	Segmentation Fault	927	Segmentation Fault
3	989	Segmentation Fault	976	Segmentation Fault	956	Segmentation Fault
4	7231	Segmentation Fault	7904	Segmentation Fault	7789	Segmentation Fault
5	8327	Segmentation Fault	7987	Segmentation Fault	7865	Segmentation Fault
6	8167	Segmentation Fault	7845	Segmentation Fault	7834	Segmentation Fault
7	7952	Segmentation Fault	7967	Segmentation Fault	7794	Segmentation Fault

Tabela 8 - Resultados obtidos nos testes realizados em Windows

A tabela 8 mostra que houve três momentos distintos na execução dos testes, tal como é possível observar pelos valores obtidos. O primeiro ensaio nas três situações de teste resultou num erro “pthread: can’t create threads”. Este erro foi facilmente resolvido uma vez cada novo *thread* que executava uma instância do *player* não estava a ser parado após a sua execução. Depois de resolvido este erro o número de sessões obtidas aumentou, no entanto o

caso de paragem era um *segmentation fault*. O código da prova de conceito foi analisado e foi detectado que o destrutor da classe document da libAmbulant não estava a ser invocado. Depois de corrigido este erro o número de sessões aumentou significativamente, rondando as oito mil sessões tal como pode ser observado pelos ensaios quatro, cinco, seis e sete. No entanto estes valores não eram os ideais, ou seja, o número de sessões atingido não era suficiente. Antes de se procurar a causa do erro passou-se esta prova de conceito para o ambiente Linux para se obter outros resultados e testar se este problema se devia apenas ao uso do ambiente Windows. Deste modo, a migração para o ambiente Linux foi feita e o código adaptado de forma a ser compatível. A tabela 9 mostra os resultados obtidos neste ambiente para os testes referidos anteriormente.

Número do Ensaio	Teste 1		Teste 2		Teste 3	
	Valor	Caso de Paragem	Valor	Caso de Paragem	Valor	Caso de Paragem
1	1237	Segmentation Fault	1211	Segmentation Fault	1187	Segmentation Fault
2	1245	Segmentation Fault	1176	Segmentation Fault	1209	Segmentation Fault
3	1321	Segmentation Fault	1168	Segmentation Fault	1185	Segmentation Fault
4	1287	Segmentation Fault	1265	Segmentation Fault	1204	Segmentation Fault
5	1216	Segmentation Fault	1198	Segmentation Fault	1192	Segmentation Fault

Tabela 9 - Resultados obtidos nos testes realizados em Linux

Os resultados obtidos em ambiente Linux não foram tão bons como os obtidos em ambiente Windows e o erro persistiu. Apesar de a lógica de execução se manter de uma versão para a outra os resultados obtidos variam, causa atribuída às diferenças nos ambientes de teste. Uma vez que o ambiente de teste usado neste de caso era Linux, foi utilizado o Valgrind[21] para depuração do código. Esta ferramenta detectou alguns *memory leaks* na prova de conceito, nomeadamente na libAmbulant, motivo pelo qual o grupo de desenvolvimento foi contactado e a situação explicada em busca de uma solução. Entretanto o desenvolvimento do interpretador não parou pelo que a

biblioteca continuou a ser utilizada, uma vez que esta cumpriu todos os requisitos excepto os de robustez.

Concluindo esta análise à prova de conceito foram retiradas as seguintes elações:

1. É possível integrar a libAmbulant noutras aplicações;
2. É possível correr um *player* SMIL sem reproduzir elementos multimédia;
3. É possível aceder à informação referente aos recursos multimédia que se encontram a ser reproduzidos quando começa a reprodução destes;
4. É possível compilar a libAmbulant com apenas as bibliotecas necessárias para o resultado pretendido.

Fica assim por provar que é possível correr várias instâncias dos *players* em simultâneo sem que haja conflitos. De uma maneira geral, as diferentes instâncias não afectam a execução das outras, havendo no entanto problemas de fuga de memória que impedem a execução repetida de cada instância, dependendo este valor da quantidade de memória disponível.

Uma vez que não é o objectivo deste projecto corrigir eventuais problemas na biblioteca, o erro foi reportado à comunidade que gere o Ambulant Player continuando-se assim o desenvolvimento do interpretador esperando-se uma solução.

5.3 Arquitectura do CSMIL

De uma forma geral, o interpretador integra-se num conjunto de módulos que operam de forma a fornecer os conteúdos a dispositivos móveis, ou seja, o CSMIL é apenas um dos componentes que integram o WSS que tem por objectivo fornecer funcionalidades avançadas na reprodução de elementos multimédia. Nesta secção, numa primeira fase, será descrito em concreto em que consiste o WSS e de que forma o módulo CSMIL se integra neste. De seguida é descrito o módulo em si, seus componentes e decisões de implementação.

5.3.1 WSS

O *Windless Streaming Server* é uma plataforma que disponibiliza conteúdos multimédia (vídeos) para, mas não limitado a, clientes de vídeo existentes em telemóveis. As plataformas que disponibilizam conteúdos multimédia para telemóveis existentes no mercado são reduzidas e na sua maioria são constituídas pela solução da Real.

As versões anteriores do WSS trabalhavam com o Darwin Streaming Server para providenciar conteúdos. Devido à necessidade de disponibilizar novas funcionalidades de manipulação de vídeo foi desenvolvida uma nova versão que utiliza hardware para poder providenciar os vídeos.

A possibilidade de controlar a apresentação de conteúdos por sessão no WSS constitui uma nova funcionalidade incluída na evolução do produto. A abordagem seguida baseia-se nos padrões desta área e resultaram na utilização da linguagem SMIL que possibilita a integração de diferentes recursos de media numa apresentação. Este módulo, designado por InoSMIL ou CSMIL, consiste num interpretador de SMIL que gere as apresentações das sessões no WSS descarregando a script de um Content Manager ou de um directório local, trocando comandos com o módulo de gestão de sessões para controlo dos recursos de média. A figura 10 pretende mostrar a arquitectura de alto nível do WSS.

Na figura 10 encontram-se diversas entidades. O Cliente é uma entidade externa que realiza pedidos de vídeo para o servidor. O Streaming Server é um componente externo que disponibiliza conteúdos de vídeo através de sessões RTSP e envia dados através via RTP.

O Session Manager é uma entidade que representa o núcleo do WSS. As suas funções incluem a gestão de todas as sessões, dos recursos de *hardware*, clientes RTSP para a disponibilização de conteúdos externos, interface com sistema de externos de gestão de negócio, controlo de distribuidores de Live Feeds…

O módulo de Monitoring é o módulo que disponibiliza estatísticas e operações de controlo sobre o WSS.

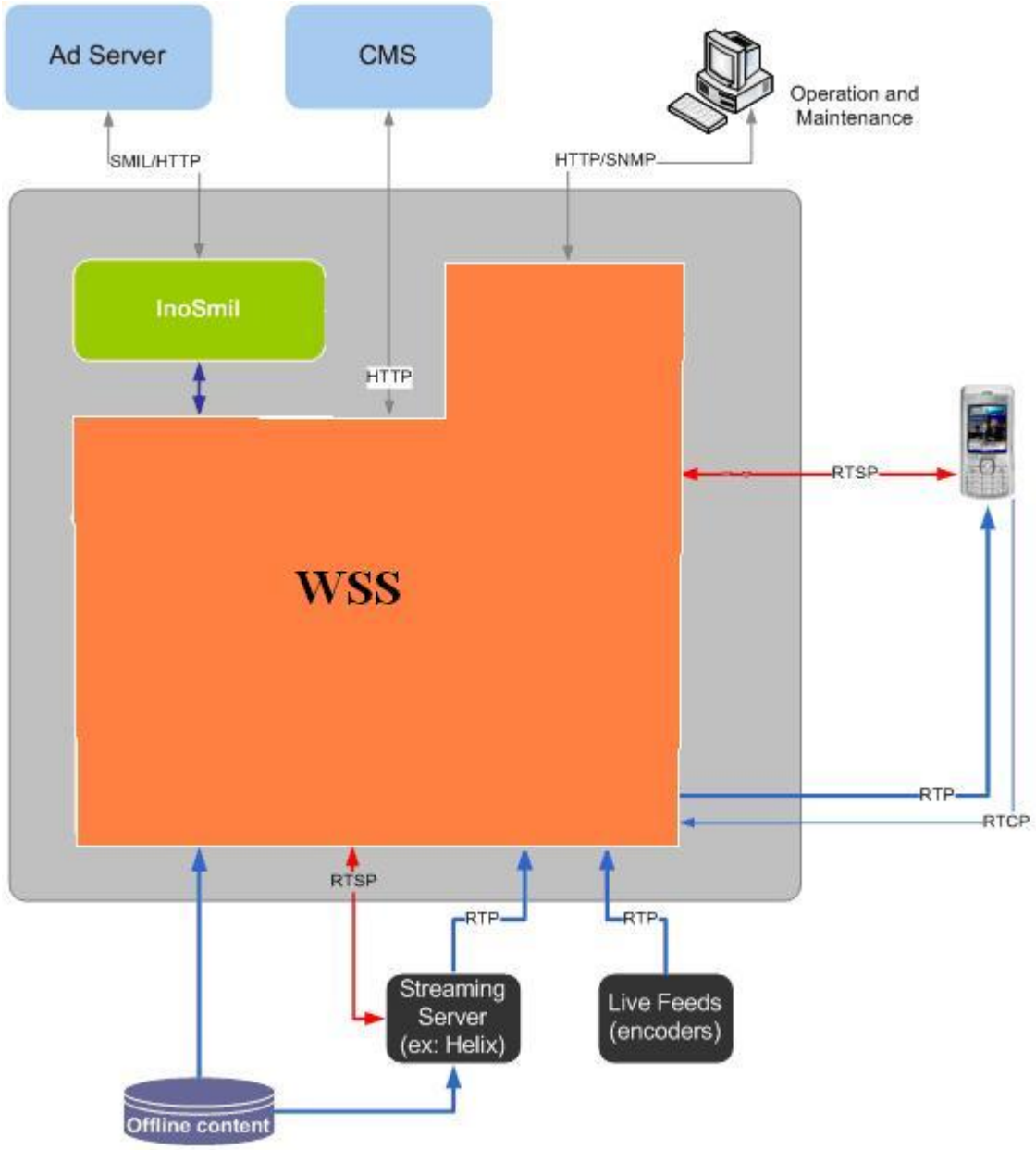

Figura 10 - Arquitectura de alto nível do WSS

A Content Management Platform (CMS) é uma plataforma externa que gere o controlo sobre o acesso a conteúdos, recebe informação de *billing* gerada, etc..

O Live Feed Distributer (LFD) é um módulo de *broadcasting* de *Live Feed* para os multiplicar por vários destinos. Este módulo permite que um *stream* único seja disponibilizado a múltiplos DSPs.

O módulo InoSmil, que é o CSMIL, é o módulo que controla a apresentação dos conteúdos multimédia, através da linguagem SMIL, por

sessão do cliente. Este módulo comunica com o Session Manager e com a Content Management Platform.

5.3.2 Protocolo de Comunicação

O interpretador de SMIL irá funcionar como uma ponte entre o servidor de conteúdos e o servidor de média, controlando e introduzindo novas funcionalidades nas apresentações de multimédia. Ele vai receber informações de controle das sessões a partir do servidor de média e documentos SMIL e informação dos elementos multimédia do servidor de conteúdos e envia pedidos de documentos SMIL e de informação sobre os recursos multimédia para o servidor de conteúdos e mensagens de controle da apresentação para o servidor de média, conforme é possível ver-se na figura 11.

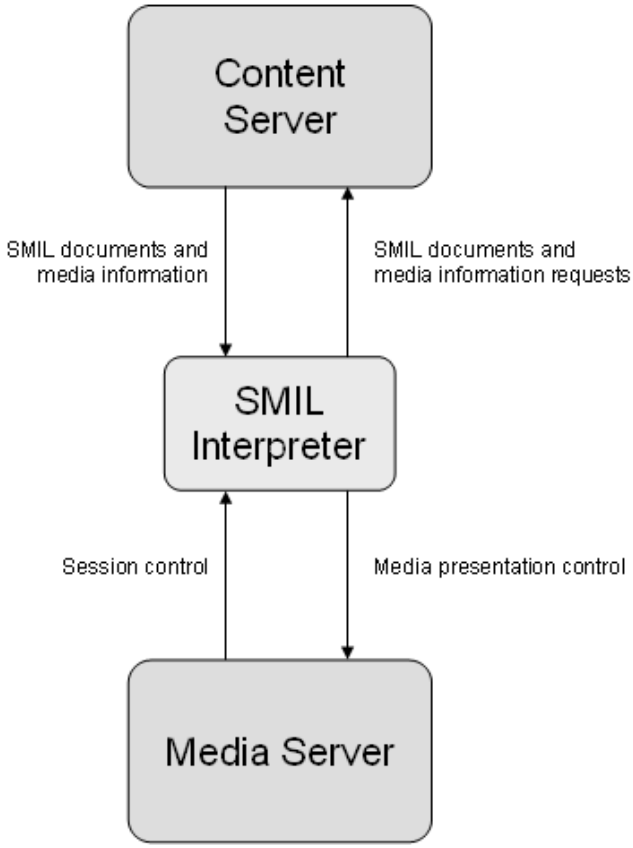

Figura 11 –Interacção do interpretador com os restantes módulos

Sendo a arquitectura do WSS modular, há necessidade de um mecanismo próprio para trocar informações entre os vários módulos de acordo com um protocolo ASCII, obedecendo a um formato proprietário, constituído por um cabeçalho e respectivo corpo da mensagem. Dado a maioria da funções serem assíncronas, existe sempre associado a uma mensagem de comando, um evento de resposta, formatado também de acordo com o mesmo protocolo. O cabeçalho contém a indicação de qual a função executada e dependendo do evento, o respectivo corpo da mensagem e baseia-se num principio de key=value para transferência de informações. Este sistema permite que o protocolo evolua conforme as necessidades.

O protocolo de comunicação entre módulos baseia-se na *layer* de comunicações LCM[18].

Todos os módulos do WSS implementam pelo menos duas mensagens de comunicação, uma de comandos e outra de *heartbeat*.

Tal como foi dito antes, as mensagens são constituídas por um cabeçalho e um corpo. A tabela 10 resume os campos do cabeçalho e explica qual a sua função.

Campo	Tipo de dados	Função
moduleID	Inteiro (16 bits)	Identificador do módulo que envia a mensagem.
sessionID	Inteiro (32 bits)	Identificador da sessão a que se destina a mensagem
timestamp	Inteiro (64 bits)	Instante temporal de geração da mensagem em milissegundos.
msgType	Inteiro (16 bits)	Comando ou evento.
errorCode	Inteiro (16 bits)	Código de erro que está associado ao evento.

Tabela 10 - Elementos do cabeçalho de uma mensagem

O campo **moduleID** guarda um identificador do módulo que envia a mensagem e da instância desse módulo que o faz. O campo **sessionID** guarda o identificador da sessão a que se destina a mensagem. O campo **timestamp** representa o instante temporal de geração da mensagem e tem como propósito a validação da mensagem, assim como assegurar que esta não se encontra fora de "tempo". O campo **msgType** define o tipo de comando ou evento e o campo **errorCode** indica se houve algum erro associado ao comando. Cada módulo define os seus códigos de erro.

Cada módulo envia também mensagens de *heartbeat* para informar o seu estado. Estas mensagens contêm os campos descritos na tabela 11.

Campo	Tipo de dados	Função
moduleID	Inteiro (16 bits)	Identificador do módulo que envia a mensagem.
timestamp	Inteiro (64 bits)	Instante temporal de geração da mensagem em milissegundos.
moduleState	Inteiro (32 bits)	Estado do módulo.
totalSessions	Inteiro (32 bits)	Número total de sessões servidas.
freeSessions	Inteiro (32 bits)	Número de sessões disponíveis.
simSessions	Inteiro (32 bits)	Número de sessões simultâneas no momento.
maxSessions	Inteiro (32 bits)	Número máximo de sessões que foram servidas em simultâneo.
timeMaxSessions	Inteiro (64 bits)	Instante temporal em que foi registado o número máximo de sessões simultâneas.
errorsCnt	Inteiro (32 bits)	Número de erros ocorridos desde que o módulo foi iniciado.
extraParams	Texto	Parâmetros extra.

Tabela 11 - Campos das mensagens de *heartbeat*

Os valores enviados nas mensagens de *heartbeat* deverão ser calculados durante a execução do módulo e enviados com estas mensagens. Sempre que no campo moduleState se encontra o valor 1 significa que o módulo se encontra sem problemas, quando se encontra o valor 0 significa que o módulo ainda não se encontra preparado para receber sessões e quando o seu valor é menos que zero significa que ocorreu um erro, onde o valor corresponde ao código do erro associado.

Existe um canal específico para as mensagens de *heartbeat* que são enviadas para todos os módulos, cabendo a cada um processar as que lhe são destinadas e descartando as demais. Os valores estatísticos destas mensagens são processados pelo módulo Session Manager que os compila para posterior apresentação numa interface Web.

Voltando às mensagens de comando, o CSMIL troca com o Session Manager catorze mensagens de comando, tal como descrito na tabela 12.

Nome	Valor	Origem	Destino	Descrição
Describe Session	201	Session Manager	CSMIL	Solicita a duração total de uma sessão.
New Session	202	Session Manager	CSMIL	Dá início a uma nova sessão
Get Media Duration	203	CSMIL	Session Manager	Solicita a duração de recursos de média.
Start Play Media	204	CSMIL	Session Manager	Pedido de reprodução de um recurso de média.
Play Media Processing	205	Session Manager	CSMIL	Resposta ao pedido de reprodução de um recurso de média.
Stop Play Media	206	CSMIL	Session Manager	Pedido de paragem da reprodução de um recurso de média.
Teardown	207	Session Manager ou CSMIL	CSMIL ou Session Manager	Sinaliza a finalização de uma sessão.
Describe Completed	208	CSMIL	Session Manager	Resposta ao pedido da duração total de uma sessão.
New Session Completed	209	CSMIL	Session Manager	Resposta ao pedido de iníciode uma nova sessão.
Get Media Duration Completed	210	Session Manager	CSMIL	Resposta ao pedido de duração de recursos de média.
Play Media Completed	211	Session Manager	CSMIL	Resposta ao pedido de reprodução de um recurso de média.
Stop Media Completed	212	Session Manager	CSMIL	Resposta ao pedido de paragem da reprodução de um recurso de média.
Teardown Completed	213	Session Manager ou CSMIL	CSMIL ou Session Manager	Resposta à sinalização do término de uma sessão.

Tabela 12 - Mensagens de comando trocadas com o CSMIL

A mensagem Describe Session encontra-se definida com o valor 201 no campo **msgType** do cabeçalho. Esta mensagem é um comando enviado pelo Session Manager que solicita ao CSMIL a duração total de uma sessão. Esta duração é calculada pelo CSMIL de acordo com a duração dos vários recursos de média que serão reproduzidos durante a sessão e de acordo com a especificação do documento SMIL. No corpo desta mensagem vem um URL com a localização do documento SMIL que será interpretado para essa sessão.

A resposta a este comando é uma mensagem de Describe Completed que assume o valor 208 no campo **msgType** do cabeçalho. No corpo da resposta segue a duração da sessão ou o valor -1 se a duração desta for indefinida, uma *flag* que indica se a sessão irá implementar a funcionalidade de *picture-in-picture* e outra *flag* que indica se a mesma utiliza endereços RTSP para indicar a localização de recursos multimédia.

A mensagem Get Media Duration solicita a duração de cada recurso de média que vão ser acedidos durante a sessão. Na mensagem são enviados um conjunto de ficheiros que serão respondidos com um valor numérico com a duração do recurso na mesma posição. O CSMIL apenas tem necessidade de pedir a duração de um recurso de média caso no documento SMIL não esteja especificado a duração deste. Em resposta a este pedido vem uma mensagem de Get Media Duration Completed.

Sempre que quer dar inícioa uma nova sessão o Session Manager envia para o CSMIL uma mensagem de New Session que tem como valor no campo **msgType** 202. Em resposta a esta mensagem o CSMIL envia uma mensagem de New Session Completed, que tem como valor no campo **msgType** do cabeçalho 209.

O comando de Teardown tanto pode ser enviado pelo Session Manager como pelo CSMIL. Este comando solicita a finalização de uma sessão e tem o valor 207 no campo msgType do cabeçalho. A resposta a este comando é a mensagem Teardown Completed. Ambas as mensagens não levam qualquer informação no seu corpo.

A mensagem Start Play Media é uma mensagem enviada pelo CSMIL para o Session Manager que corresponde um pedido para reproduzir um recurso de media. No corpo desta mensagem vão as seguintes informações:

1. Tipo de recurso que se pretende que seja reproduzido, ou seja, se é uma imagem ou logótipo, se é um vídeo ou se é um texto;
2. Um identificador do recurso multimédia;
3. O URL do recurso que se pretende que seja reproduzido ou o texto que se pretende mostrar;

As seguintes informações são opcionais e seguem no corpo da mensagem também caso estejam definidas no documento SMIL:

1. A distância ao topo da janela, ou seja, a posição do recurso em pixéis relativamente ao topo da janela (esta informação é extraída das regiões definidas no documento SMIL).
2. A distância relativa à margem esquerda da janela, também extraída das regiões do documento.
3. Largura da região onde será reproduzido o recurso (válido apenas para vídeos e logótipos).
4. Altura da região onde será reproduzido o recurso (válido apenas para vídeos e logótipos).
5. Tamanho da fonte do texto (válido apenas para texto).
6. Cor de fundo (válido apenas para texto).
7. Estilo da fonte (válido apenas para texto).
8. Cor (válido apenas para texto).
9. Direcção do texto (válido apenas para texto).
10. ZOrder ou ordem relativamente ao topo da janela pelo qual será reproduzido o recurso (válido apenas para vídeo).

Em resposta a este comando vem uma mensagem de Play Media Processing (código 205) caso o recurso já se encontre a ser reproduzido ou uma mensagem de Play Media Completed (código 211) caso o início da reprodução do recurso tenha sido efectuada com sucesso.

Sempre que chega a altura de parar de reproduzir um recurso o CSMIL envia para o Session Manager uma mensagem de Stop Play Media (código 206) e no corpo da mensagem leva o identificador do recurso que se encontra a ser reproduzido. O CSMIL espera por uma mensagem de Stop Media Completed do Session Manager como resposta para poder continuar a reprodução do documento SMIL.

Até à data o Content Server ainda não se encontrava disponível, assim como o protocolo de comunicação com este, motivo pelo qual o CSMIL apenas interpreta documentos SMIL que se encontram definidos numa localização em disco, especificada pelo Session Manager.

5.3.3 Lógica de Processamento de Uma Sessão

Desde que o módulo de *signaling* recebe um pedido para uma nova sessão no servidor de *streaming* uma série de passos são executados até o término desta. A figura 12 resume todos os passos a nível de comunicação e processamento destes acontecimentos.

Numa primeira fase é processada toda a informação contida no documento SMIL sobre a sessão. O Session Manager envia um pedido de descrição da sessão e o interpretador SMIL vai buscar e faz o *parsing* do documento. Tal como descrito da figura 13, depois de recebido o endereço do documento que se pretende interpretar, se este existir é carregado, caso contrário é carregado o documento por defeito que se encontra definido num ficheiro de configuração. De seguida o documento é processado e calculada a sua duração, se utiliza a funcionalidade de *picture-in-picture* e se usa endereços RTSP. Caso seja necessário pedir a duração de algum recurso, o interpretador percorre todo o documento à procura de eventuais recursos cuja duração seja necessária e pede-a ao Session Manager, para posteriormente calcular a duração do documento.

Depois de enviada a duração da sessão e restantes informações relevantes para o Session Manager, o interpretador espera que este dê ordem para o início da reprodução da sessão. Quando esta é recebida é chamado o método **start** do *player* de SMIL da libAmbulant. Este dá inícioà reprodução do documento, agendando os *timings* para a reprodução de cada recurso, enviando comandos de *start* e *stop* aquando do início ou término da reprodução de um recurso. Dependendo do documento, pode-se encontrar um ciclo no qual são agendados e enviados vários comandos de *start* e *stop*. Depois do envio de um comando de *stop* o *player* fica em pausa até que o Session Manager sinalize que o recurso foi efectivamente parado. Depois de reproduzido todo o documento o interpretador envia um comando de *teardown* para o Session Manager e posteriormente termina a sessão. A cada momento no decorrer da sessão o Session Manager pode enviar um comando de *teardown* para o interpretador, dando ordem para que este termine a sessão.

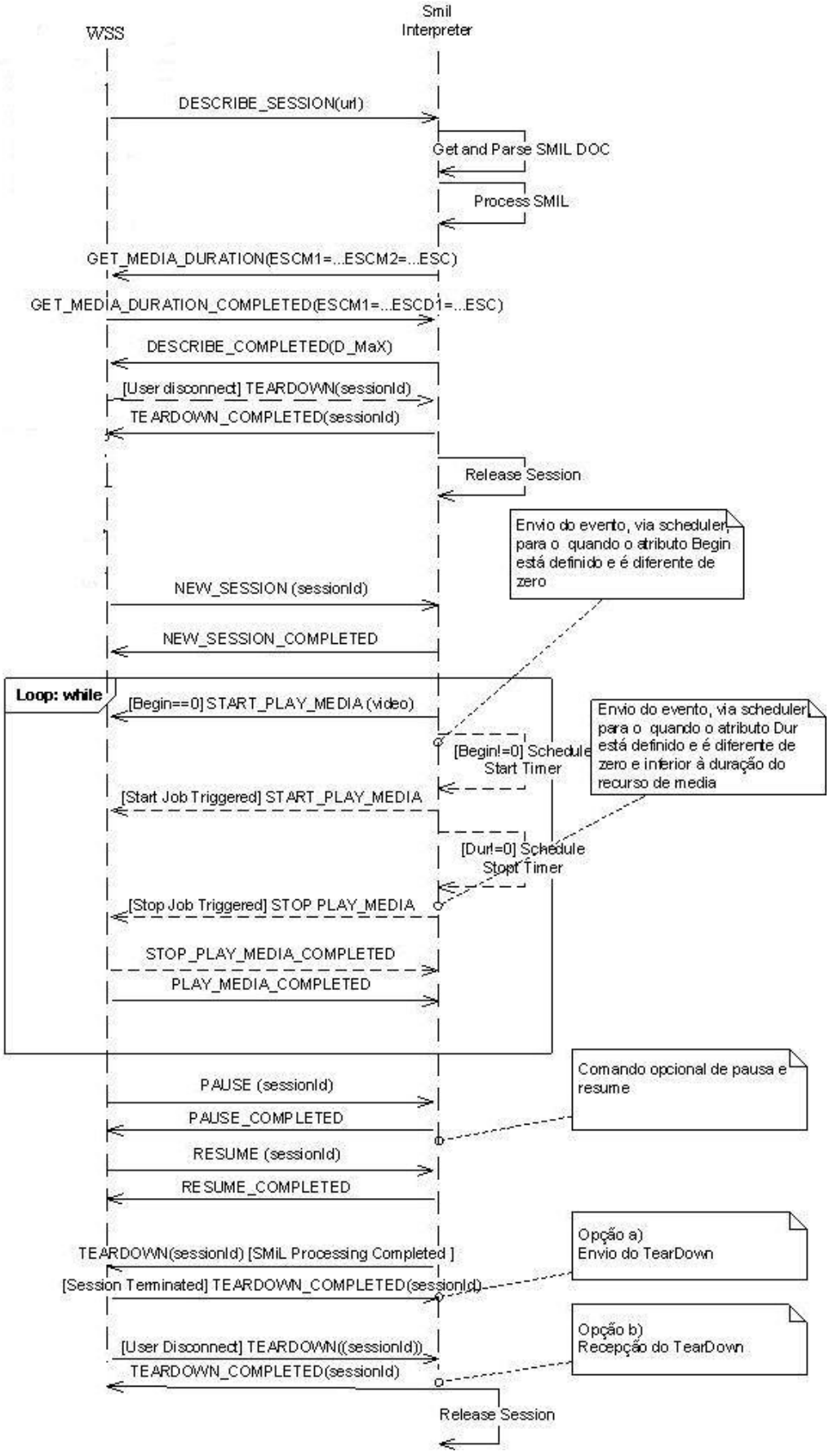

Figura 12 - Interacções externas do interpretador SMIL e lógica de funcionamento

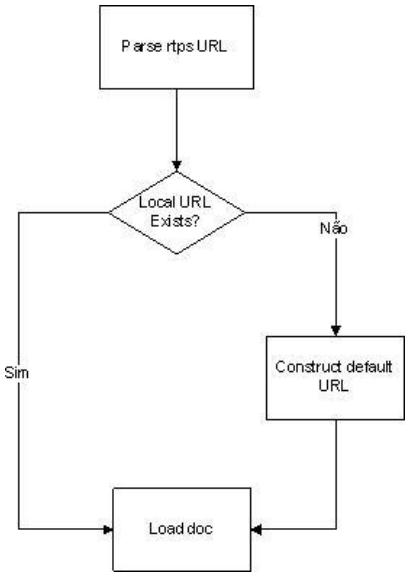

Figura 13 - Lógica de carregamento de uma sessão

Deste modo, uma sessão tem vários estados que podem ser consultados na figura 14. Podemos considerar cinco estados para uma sessão:

1. Resposta a um pedido de descrição: Neste estado, o interpretador recebe o endereço do documento SMIL que será reproduzido na sessão. Faz o *parsing* e processa este documento. Inicializa o SMIL *player* da libAmbulant que será usado para a sessão. Verifica se consegue calcular a duração total da sessão sem pedir nenhuma duração de algum recurso. Se for possível envia uma mensagem com as informações para o Session Manager e passa para o estado de nova sessão, caso contrário pede as respectivas durações e passa para o estado de processamento da duração de recursos.
2. Processamento da duração de recursos: Neste estado o interpretador recebe uma lista com as durações dos recursos cuja duração não consegui calcular, actualiza o documento SMIL com as durações em falta, calcula a duração final da sessão e envia as informações para o Session Manager e passa para o estado de nova sessão.
3. Nova sessão: Neste estado é enviada uma mensagem ao Session Manager indicando que o interpretador se encontra pronto para iniciar a reprodução do documento e é invocado o start do *player* de SMIL da libAmbulant. É agendado o início e o fim da reprodução de cada recurso e a sessão passa para o próximo estado, o estado de reprodução dos recursos.

4. Reprodução dos recursos: Neste estado, sempre que chega o momento de envio de uma mensagem indicando o inícioda reprodução de um elemento este procedimento é executado. Sempre que chega ao instante de parar a reprodução de um recurso é sinalizado o evento e esperada uma resposta indicando que o recurso foi parado. Depois de reproduzidos todos os recursos e chegando ao fim da sessão a sessão avança para o estado seguinte, o de término da sessão.
5. Término da sessão: Neste estado é terminada a sessão e os recursos são libertados. A *thread* onde corre a instância do SMIL *player* é destruída e os comandos de teardown são enviados.

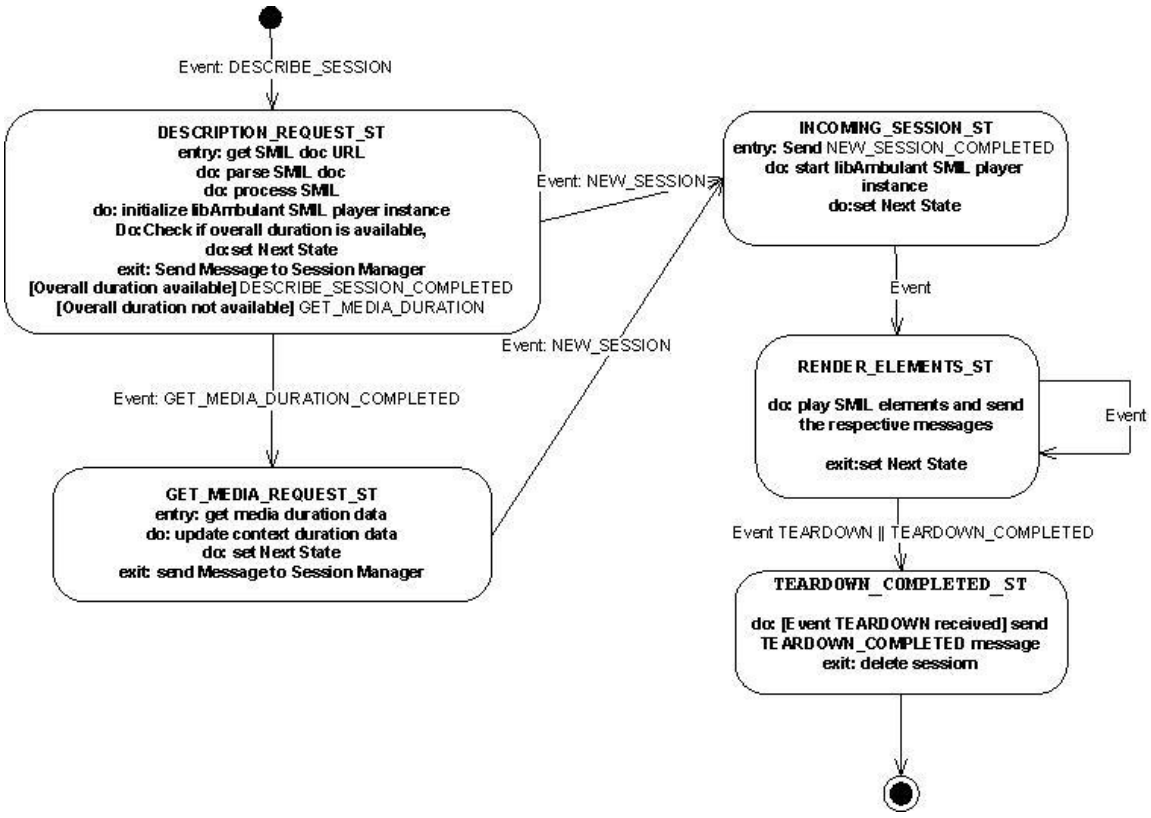

Figura 14 - Máquina de estados de uma sessão SMIL

5.3.4 Componentes do CSMIL

No que diz respeito à arquitectura do CSMIL, esta pode ser dividida em quatro elementos distintos e responsáveis por diferentes tarefas. Estes elementos, ou componentes, são:

1. O módulo de comunicação – Módulo responsável pela recepção e envio de mensagens para os restantes módulos;
2. O módulo de gestão de sessões – Módulo responsável por controlar cada sessão, lançar novas sessões e controlar o número de sessões simultâneas;
3. O módulo de *logging* – Módulo que implementa um logger para registo de informações para o cálculo de estatísticas e depuração da execução do módulo;
4. Os SMIL Players – Entidades que representam uma instância de um *player* básico da libAmbulant.

A figura 15 indica de que forma estes quatro componentes se interligam.

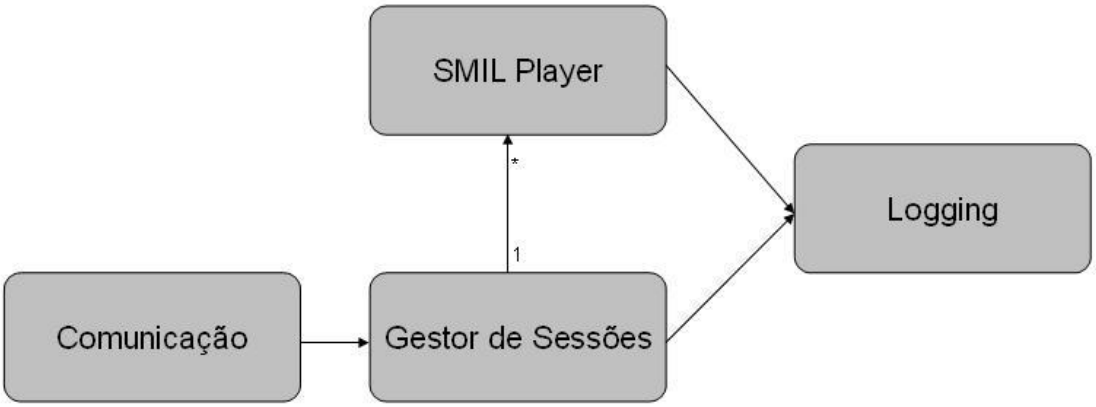

Figura 15 - Relação entre os componentes do CSMIL

O módulo de comunicação é composto por duas classes, uma responsável pelo envio e recepção de mensagens e de *heartbeats* e outra que representa uma mensagem. A figura 16 apresenta um diagrama de classes desta componente.

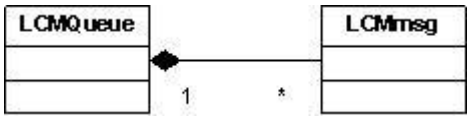

Figura 16 - Diagrama de classes do módulo de comunicação

Tal como se pode ver pela figura 16, numa instância da **LCMQueue** estão presentes várias mensagens (**LCMmsg**). A classe **LCMmsg** apresenta uma abstracção da uma mensagem e a **LCMQueue** apresenta uma fila de armazenamento das mensagens chegadas e processa o envio de mensagens. Esta última processa ainda a validação de *heartbeats*.

A classe **LCMQueue** implementa uma instância da biblioteca LCM para envio e troca de mensagens. São definidos quatro canais de comunicação:

1. Um canal para recepção de mensagens de comando;
2. Um canal para envio de mensagens de comando;
3. Um canal para a recepção de *heartbeats*;
4. Um canal para o envio de *heartbeats*;

Todas as mensagens de comandos que chegam ao CSMIL são armazenadas numa fila de mensagens e a cada ciclo de processamento é retirada da fila a primeira mensagem e processada. O LCM permite definir uma função (*callback*) que é chamada sempre que é recebida uma mensagem num determinado canal. Sempre que esta função é chamada porque é recebida uma mensagem de comando, esta mensagem é introduzida no fim da fila. No caso dos *heartbeats*, sempre que um é recebido é assinalada uma *flag* que indica que tudo se encontra ou, em caso de erro, qual o motivo. O envio de mensagens de *heartbeat* é directo, ou seja, sempre que tal é necessário o gestor de sessões indica à **LCMQueue** para o fazer, passando os parâmetros que devem ser enviados com esta mensagem. A instância do **LCMQueue** corre numa *thread* independente pelo que a recepção de mensagens é contínua.

No que toca ao módulo de *Logging*, este é composto por apenas uma classe, tal como é visível no seu diagrama de classes (Figura 17).

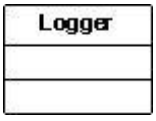

Figura 17 - Diagrama de classes da componente de *logging*

Para a funcionalidade de *logging* foi utilizada uma biblioteca da PT Inovação chamada Inolog. Esta biblioteca utiliza o rsyslog[22] para o processamento dos logs. A classe **Logger** não faz mais que criar uma interface do sistema (escrito em C++) com a biblioteca Inolog (escrita em C), utilizando as funcionalidades deste. O Inolog aceita sete níveis de *logging* que se encontram descritos na tabela 13.

Nível	Descrição	Tipo de acontecimento
0	Emergency	O sistema não se encontra usável.
1	Alert	Uma acção tem que ser tomada imediatamente.
2	Critical	Aconteceu uma condição crítica.
3	Error	Condição de erro.
4	Warning	Condição de aviso.
5	Notice	Acontecimento normal mas significante.
6	Info	Informacional
7	Debug	Mensagem de debug.

Tabela 13 - Níveis de *logging*

O ficheiro apresentado na figura 18 é um exemplo de uma de um ficheiro de *log*, gerado no decurso de um dos testes realizados.

```
2009 Oct 12 20:56:54.358 INF  csmil 26086: Describe Session received.
Id=33620202

2009 Oct 12 20:56:54.358 DBG  csmil 26086: Parsed SMIL document URL.
Doc="/var/ptin/wss/media/ex1.xml". sID=33620202.

2009 Oct 12 20:56:54.358 NOT  csmil 26086: NSessSimult =77 .
NTotalSess = 37517 .

2009 Oct 12 20:56:54.358 INF  csmil 26086: New SMIL player instance
created. Id=33620202

2009 Oct 12 20:56:54.358 INF  csmil 26086: Started SMIL player
instance. Id=33620202

2009 Oct 12 20:56:54.358 INF  csmil 14369: Document loaded.
URL="/var/ptin/wss/media/ex1.xml". Id=33620202

2009 Oct 12 20:56:54.359 INF  csmil 14369: Calculating document
duration. SessionID=33620202
```

Figura 18 - Parte de um ficheiro de log

Neste exemplo não existe nenhuma situação de erro registada, no entanto, como exemplo de situações de erro temos casos em que o documento SMIL não se encontra de acordo com a norma, ou seja, quando o *parsing* deste

não é processado correctamente (nível 3), ou situações em que exista um *timeout* na comunicação com o Session Manager (nível 1).

No que toca às instâncias dos *players*, estas são responsáveis pelo controlo da reprodução dos recursos de multimédia de cada sessão. Cada instância representa uma instância da libAmbulant, ou seja, de um *player* SMIL. Houve necessidade de alterar a arquitectura do Ambulant Player de forma a que os objectivos fossem cumpridos. A primeira grande alteração foi retirar todos os *renderes* de media, uma vez que não se pretende reproduzir os recursos mas sim enviar mensagens de início e fim da sua reprodução. Tal como havia sido testado com a prova de conceito, esta alteração é fácil de executar, bastando para tal não instanciar a Datasource Factory, levando a que a biblioteca tenha o seu comportamento por defeito que é ignorar a reprodução dos recursos.

Como são enviadas mensagens ao Session Manager indicando o início e o término da reprodução dos elementos de media, é necessário criar uma nova instância da Playable Factory. Esta nova instância envia uma mensagem de início da reprodução de um recurso sempre que o nó referente a esse recurso é activado e uma mensagem de término sempre que esse nó se torna inactivo. Esta alteração do estado de cada nó é controlada pelo *player* de acordo com a linha de tempo definida no seu *scheduler*. Sempre que um nó se torna activo pelo *player* é invocado um *callback* na Playable Factory chamado de **start** e sempre que este se torna inactivo é invocado o *callback* de stop. Esta alteração efectuada ao normal funcionamento da biblioteca também podia ter sido feita de uma outra maneira. Em vez de se retirar a Datasource Factory, poderia-se criar uma nova versão desta que em vez de fazer o *render* dos recursos enviaria as mensagens. No entanto, esta seria uma má opção uma vez que o Session Manager necessita de informações adicionais sobre cada elemento multimédia e estas não se encontram disponíveis na Datasource Factory. O Session Manager tem recursos limitados para cada sessão pelo que quando se envia uma mensagem indicando o término da reprodução de um elemento, toda a sessão fica parada à espera que este envie uma mensagem indicando que o recurso foi efectivamente parado de modo a não haver conflitos com os restantes elementos reproduzidos.

Uma terceira alteração foi efectuada ao normal funcionamento da biblioteca. Quando se usa o *player* básico da biblioteca, este tem definido a uma classe própria chamada de embedder. Esta classe guarda *callbacks* que são invocados sempre que o *player* carrega um documento, que a reprodução de um documento termina, quando fica em estado de pausa, etc.. Para o CSMIL apenas interessa o *callback* que é chamado quando termina a reprodução de um documento para libertar os recursos e enviar uma mensagem de *teardown* ao Session Manager. Desta forma, a classe Embedder também foi alterada.

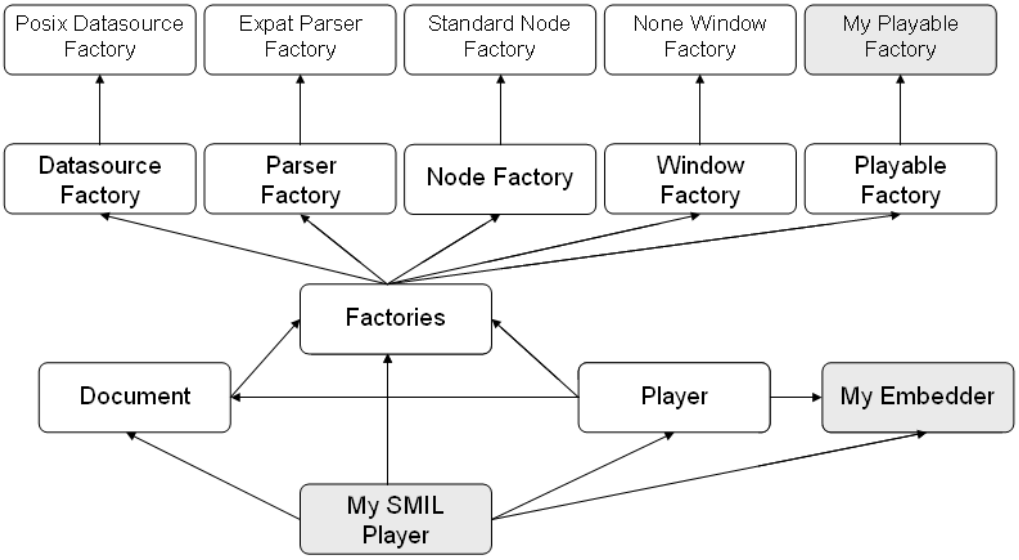

Figura 19 - Componentes da libAmbulant e alterações efectuadas

A figura 19 mostra os diferentes componentes da biblioteca e quais os que foram substituídos de forma a cumprir o objectivo do interpretador, a cinzento. Uma vez que se pretende que o interpretador tenha elevado desempenho, parte-se do princípio que os documentos SMIL que serão interpretados encontram-se válidos, motivo pelo qual é sempre usado o Expat para fazer o *parsing* deste, através da Expat Parser Factory. Na Datasource Factory apenas é carregada a Posix Datasource Factory pois esta é usada para abrir ficheiros nos sistemas operativos Unix. A Window Factory é usada pela biblioteca para criar novas janelas mas, uma vez que o interpretador não reproduz elementos multimédia, esta não é inicializada assumindo o seu valor padrão que é None Window Factory, que não implementa a funcionalidade

desejada. Desta forma, cada instância do *player* apresenta o diagrama de classes apresentado na figura 20.

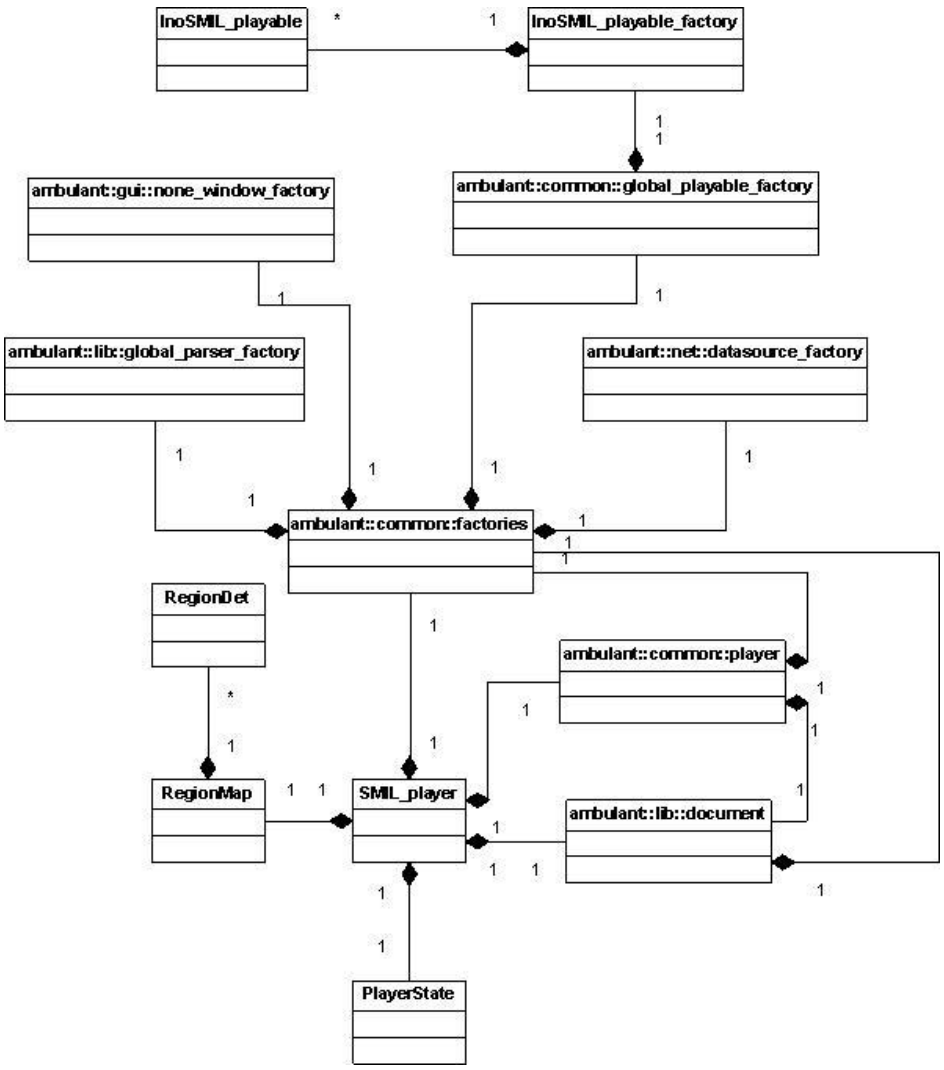

Figura 20 - Diagrama de classes de uma instância do *player*

Neste diagrama são visíveis três classes cuja função não foi referida anteriormente. Uma dessas classes é a PlayerState, que serve para guardar a máquina de estados do SMIL_player. A classe RegionDet guarda os detalhes de uma região e permite algumas operações sobre estas, como alterar o seu nome ou alterar alguns detalhes desta. A classe RegionMap contém um mapa de todas as regiões do documento que é preenchido a quando do processamento do deste para o cálculo da sua duração final.

O módulo Gestor de Sessões é composto apenas por duas classes, uma primeira que guarda a configuração inicial do interpretador e outra onde são

realizadas as operações de novas sessões, descrição de sessões, etc.. A figura 21 mostra o diagrama de classes deste módulo.

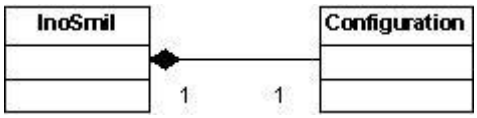

Figura 21 - Diagrama de classes da componente de gestão de sessões

A classe InoSmil faz todas as tarefas de gestão do interpretador, desde controlar a monitoria dos *heartbeats*, a consulta da fila de mensagens, o processamento de novas sessões, resposta a pedidos de descrição de uma sessão, término de sessões, inicialização das classes que interagem com este módulo. Tal como foi dito anteriormente, o interpretador quando arranca lê um conjunto de configurações a partir de um ficheiro, que são guardadas na classe Configuration. A figura 22 mostra um exemplo de um ficheiro de configuração.

Tal como podemos ver pela figura 22, o ficheiro de configuração permite definir os parâmetros descritos na tabela 14.

Parâmetro	Descrição
log.csmil.all	Parâmetro de log.
log.csmil.finest	Parâmetro de log.
log.csmil.facility	Parâmetro de log.
log.csmil.name	Parâmetro de log.
log.csmil.output	Parâmetro de log.
log.csmil.namelen	Parâmetro de log.
log.csmil.filesize	Parâmetro de log.
MaxSessions	Número máximo de sessões em simultâneo.
MaxNewSessions	Número máximo de novas sessões por segundo.
SendHeartBeat	Período de envio de heartbeat para o Session Manager em segundos.
RecHeartBeat	Período de recepção de heartbeat do Session Manager em segundos.
MoviesPath	Caminho absoluto para o directório dos vídeos.
TUnit	Unidade de tempo usada para as comunicações com o Session Manager (segundos ou milissegundos).
SMPath	URL do *layer* de comunicação entre módulos.

Tabela 14 - Parâmetros definíveis por ficheiro de configuração

```
[Logging]

log.csmil.all=1
log.csmil.finest=0
log.csmil.facility=local0
log.csmil.name=csmil
log.csmil.output=file:/opt/ptin/wss/logs/wss-csmil.log
log.csmil.namelen=6
log.csmil.tidlen=5
log.csmil.filesize=500000

[SMILInterpreter]

# Maximum number of sessions
# Format: Integer - [1 : 2000]
# Default: 500
MaxSessions = 500

# Maximum number of new sessions
# Format: Integer - [1 : 100]
# Default: 50
MaxNewSessions = 50

# Send Heartbeat interval
# Format: Integer - [1 : 10]
# Default: 5
SendHeartBeat = 5

# Receive Heartbeat interval
# Format: Integer - [1 : 10]
# Default: 5
RecHeartBeat = 5

# Movies path
# Format: String path without '/' in the end
# Default: ""
MoviesPath = /usr/local/movies

# Time Unit
# Format: "sec" | "msec"
# Default: "msec"
TUnit = msec

# LCM SessionManager path
# Format: String path
# Default: ""
SMPath = udpm://239.255.76.68:7668?ttl=0&rec_buf_size=2097152
```

Figura 22 - Diagrama de classes da componente de gestão de sessões

Estes parâmetros são passados na criação de cada classe que deles necessita e apenas são carregados no arranque do interpretador, não podendo ser alterados durante a execução deste.

5.3.5 Estrutura Algorítmica do CSMIL

A lógica de funcionamento do CSMIL assenta na utilização de um *main loop* no qual são processados os pedidos enviados pelo Session Manager. Tal como foi dito na descrição do módulo de comunicação, as mensagens de comandos enviadas pelo Session Manager são armazenadas numa fila. Deste modo, o ciclo principal, de uma maneira muito geral, não faz mais que retirar a primeira mensagem da fila, processa-la e repetir este processo enquanto o módulo estiver activo. A figura 23 resume o funcionamento do *main loop*.

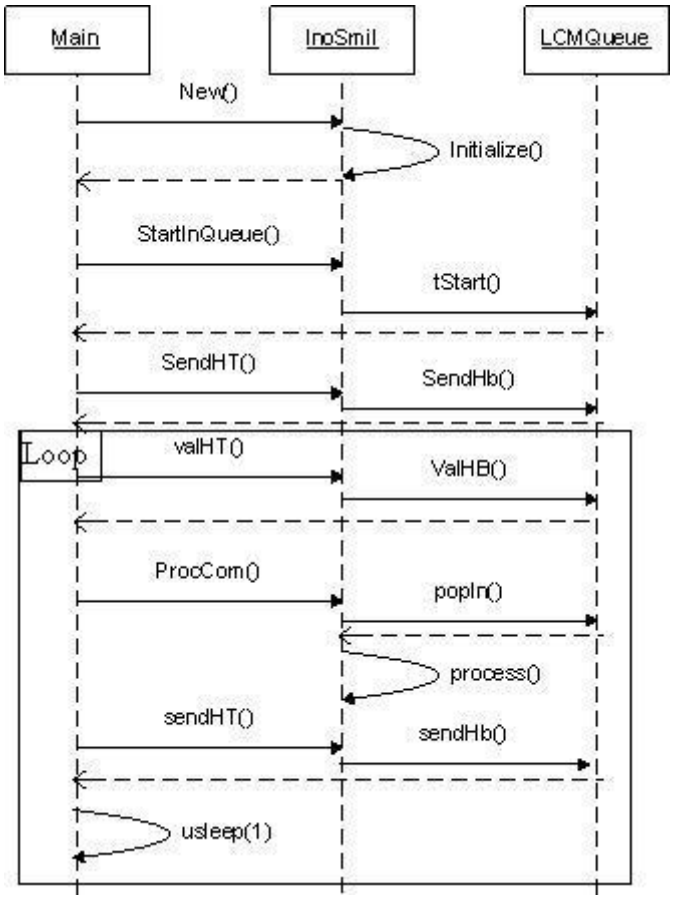

Figura 23 - Diagrama de sequência do *main loop* do CSMIL

O primeiro passo no arranque do interpretador é criar uma nova instância da classe **InoSmil**, classe que representa o gestor de sessões. Quando esta instância é criada, esta inicializa-se, ou seja, carrega as configurações a partir do ficheiro de configuração, inicializa o módulo de comunicação e o de *logging* e inicializa as variáveis que guardam as informações estatísticas. No entanto, neste ponto, o módulo de comunicação ainda não aceita nenhuma mensagem

de comando, nem *heartbeat* do Session Manager. Para tal é necessário invocar o método **startInQueue()** do **InoSmil**, para que o módulo de comunicação corra num novo *thread* e comece a aceitar mensagens. De seguida é enviado um *heartbeat* para o Session Manager para que este perceba que o CSMIL se encontra *online*. Depois da sinalização de que o interpretador se encontra activo começa o ciclo principal deste, começando pela validação dos *heartbeats* recebidos, seguindo-se do processamento do primeiro elemento da fila de comandos e terminando com o envio de um *heartbeat.* Na validação dos *heartbeats* recebidos, o CSMIL valida o espaço de tempo decorrido desde a recepção do último *heartbeat*. Se o intervalo decorrido for igual ou menor ao intervalo definido no ficheiro de configuração o método retorna o valor verdadeiro e o ciclo principal prossegue, caso contrário o interpretador para de aceitar novas mensagens de comando na fila e para todas as sessões, destruindo depois todas as instâncias. No processamento do primeiro elemento da fila de mensagens de comando, é retirado o seu primeiro elemento através do método **popIn()**, que retorna a mensagem e retira-a da fila. De seguida é analisado o tipo de mensagem recebida e invocado o método respectivo de processamento da mensagem. Quanto ao envio do *heartbeat*, é invocado o método **sendHT()**, que não faz mais que verificar se já passou o intervalo de tempo definido no ficheiro de configuração para o envio de *heartbeats*. Em caso afirmativo é enviado um novo *heartbeat* e registado esse instante. Em caso negativo volta-se para o ciclo principal da aplicação. Depois de efectuados estes três passos é feito um **usleep(1)**, ou seja, a aplicação pára durante um microssegundo para que este ciclo não consuma todos os recursos de processamento da máquina.

Todo o processamento de pedidos de novas sessões, início de sessões ou término de sessões processa-se agora a partir do método **procCom()** que retira o primeiro elemento da fila de recepção de comandos e invoca o respectivo método de processamento da mensagem. Assim, temos o caso em que a mensagem de comando enviada é um pedido de descrição de uma sessão. A figura 24 apresenta o diagrama de sequência que ilustra este caso.

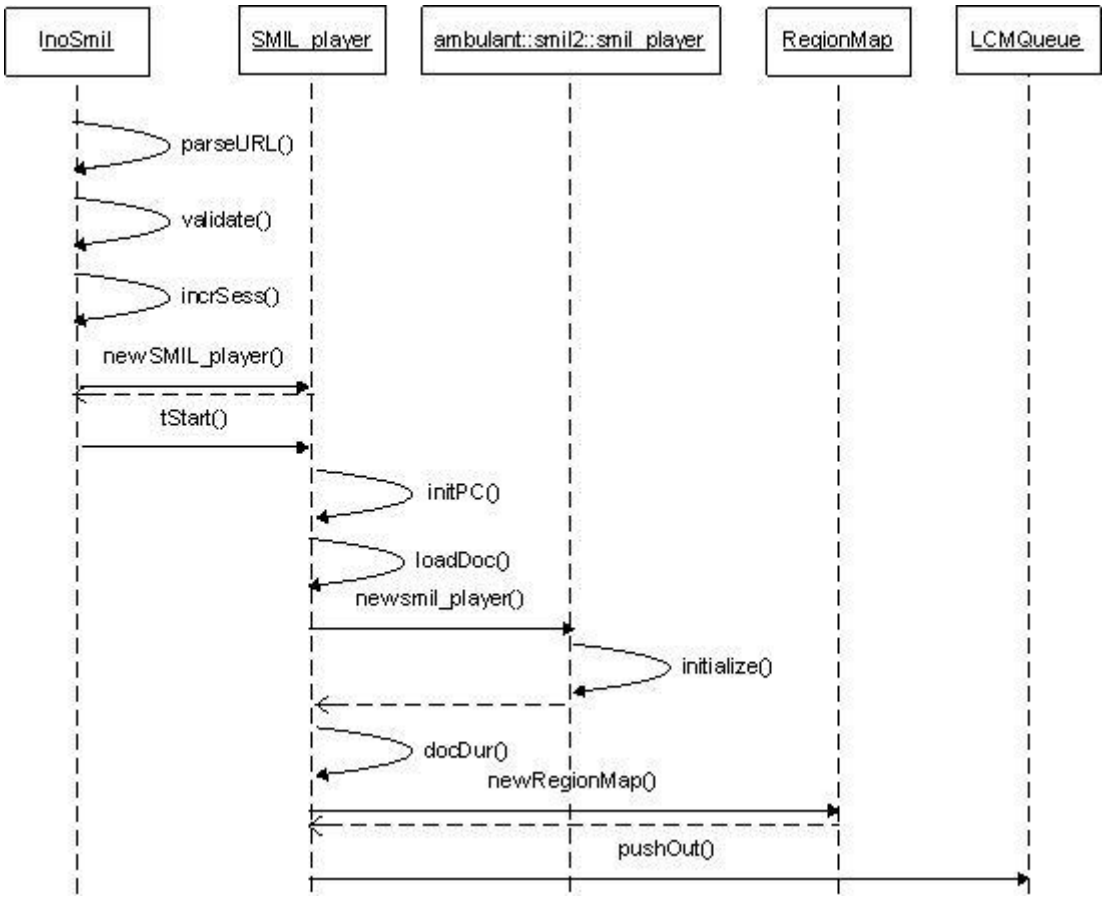

Figura 24 - Diagrama de sequência do processamento de um pedido de descrição de sessão

Quando a mensagem a ser processada é um pedido de descrição de uma sessão é invocado o método **procDS()** da classe **InoSmil**. Este método, numa primeira fase, processa o URL enviado no pedido e verifica se este aponta para um ficheiro existente no sistema. Caso este não exista, é carregado um ficheiro definido por defeito. De seguida, verifica-se se é possível instanciar a nova sessão, ou seja, é verificado se com esta nova sessão não é ultrapassado o limite de novas sessões por segundo nem o número máximo de sessões em simultâneo. O passo que se segue é verificar se não se está num novo máximo de sessões a serem processadas e se for este caso são guardados o máximo de sessões e o instante em que ocorreu. O número de sessões existentes no interpretador também é actualizado. De seguida o identificador de sessão é registado e é criada uma nova instância do SMIL_player. Depois de criada a nova instancia do **SMIL_player**, é invocado o método **tStart()** deste que arranca a instância num novo *thread,* começando por inicializar a instância e todas as *factories* necessárias para esta. De seguida é carregado o documento SMIL que foi passado na criação do **SMIL_player** e é

criado um novo **smil_player** da biblioteca do Ambulant, seguido da sua inicialização. Neste ponto é percorrida toda a árvore do documento SMIL gerada aquando do carregamento do documento para o cálculo da duração da sessão, análise de casos de *picture-in-picture* e de uso de endereços RTSP e para o registo das definições de regiões. Depois de analisados estes aspectos, caso seja possível calcular a duração final da sessão sem pedir a duração de nenhum recurso, é enviada uma mensagem de resposta ao pedido de descrição de sessão com as respectivas informações. Caso não seja possível, é enviada uma mensagem com o pedido das durações dos recursos necessários.

Quando é necessário informações sobre a duração de algum recurso para o cálculo da duração de uma sessão, o Session Manager envia uma resposta a este pedido que entra na fila de mensagens de comando recebidas. Quando o **InoSmil** encontra uma mensagem deste tipo invoca o método **procMDC()**. A figura 25 resume de que forma este comando é processado.

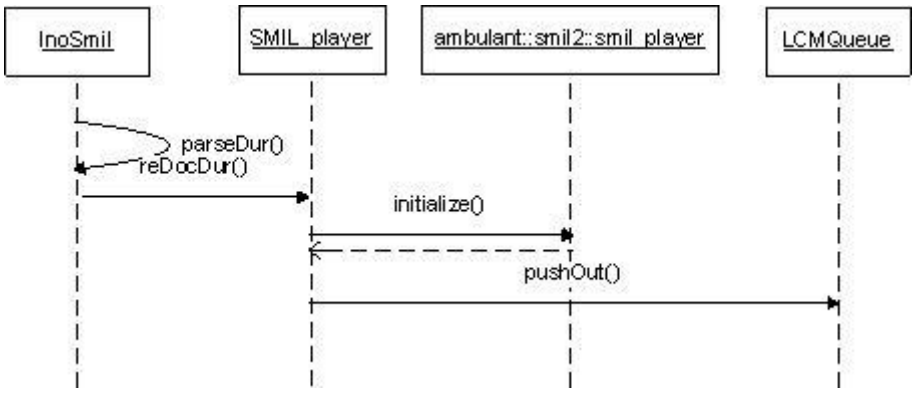

Figura 25 - Diagrama de sequência do processamento de uma mensagem com a duração de recursos multimédia

Este método primeiro processa as durações recebidas e cria um mapa de durações. De seguida procura a sessão a que as durações se destinam e invoca o método **reDocDur()** da respectiva sessão, passando o mapa. Este método volta a percorrer toda a árvore do documento calculando a sua duração final e definindo a duração dos elementos cuja duração não se encontra definida. De seguida, é reinicializado o **smil_player** da biblioteca do Ambulant e é enviada a mensagem de resposta ao pedido de descrição da sessão para o Session Manager com as informações necessárias.

Quando o Session Manager pretende dar início à reprodução de uma sessão, este envia uma mensagem de comando que é armazenada na fila de recepção de comandos. A figura 26 ilustra o que acontece no processamento destas mensagens.

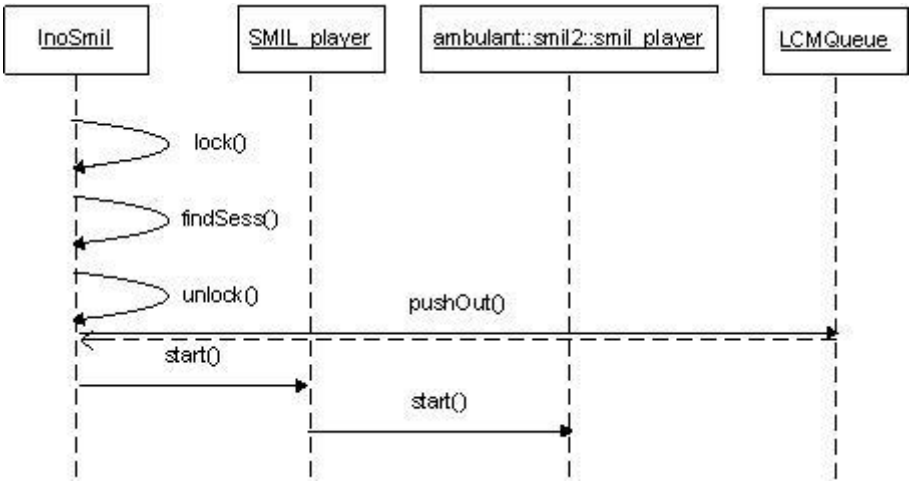

Figura 26 - Diagrama de sequência do processamento de uma mensagem de início de reprodução

Quando o **InoSmil** faz **popIn()** de uma mensagem de início da reprodução de uma sessão é invocado o método **procNS()** que processa este tipo de mensagens. Este método começa por adquirir um *lock* no mutex do **InoSmil** para que mais nenhuma acção seja efectuada na sessão enquanto esta não começar a ser reproduzida. De seguida é procurada a sessão, o mutex é libertado, é enviada uma resposta ao Session Manager. De seguida é chamado o método **start** do **SMIL_player** que invoca o método **start** do **smil_player** da libAmbulant. Este último método inicia a reprodução do documento conforme descrito na execução do Ambulant Player.

A qualquer momento no decurso da reprodução de uma sessão, esta pode receber um comando de *teardown* do Session Manager. Este comando, tal como todos os outros, é armazenado na fila de recepção de mensagens e quando o **InoSmil** faz **popIn** deste invoca o método **procTD()** que processa este tipo de mensagens. Tal como podemos ver pela figura 27, este método começa por procurar a sessão a que se refere o pedido. De seguida define o estado de pedido de *teardown* do Session Manager na máquina de estados do **SMIL_player**, evitando que este, aquando da paragem dos recursos, fique à espera de uma mensagem de stop media completed. Após a definição do estado, é invocado o método **stop** do **SMIL_player**, que invoca o método **stop**

do **smil_player** da libAmbulant. Seguidamente, é invocado o destrutor da classe SMIL_player, que destrói todas as variáveis usadas, liberta os recursos de memória ocupados e destrói o *thread* onde se encontrava a correr o **SMIL_player**. De seguida é enviada a mensagem de resposta ao Session Manager indicando que a sessão foi terminada com sucesso.

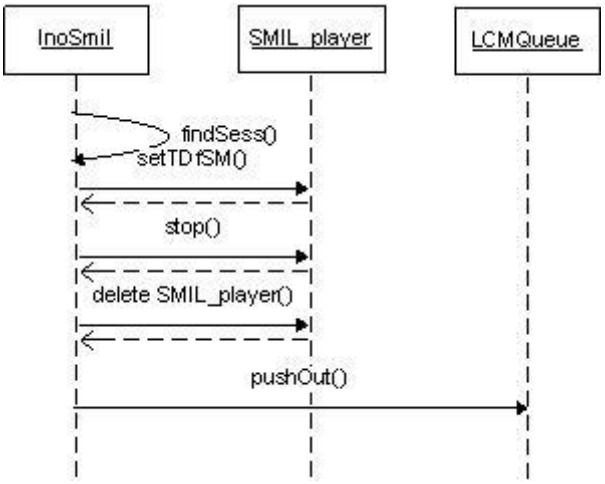

Figura 27 - Diagrama de sequência do processamento de um pedido de *teardown*

Quando o **SMIL_player** termina a reprodução de um documento envia uma mensagem de *teardown* para o Session Manager. A figura 28 ilustra o que acontece nesta situação.

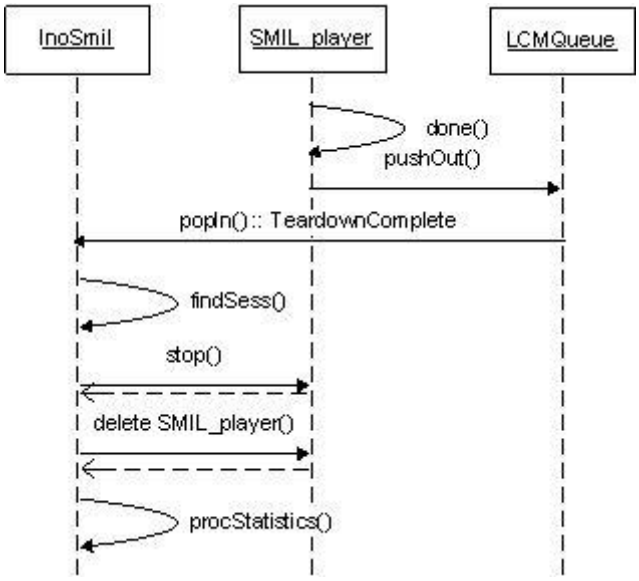

Figura 28 - Diagrama de sequência do término de uma sessão

Quando termina a reprodução de um documento, o **smil_player** invoca o *callback* **done()** do embedder. Como o embedder do **smil_player** encontra-se definido na classe **SMIL_player**, é o método **done** desta que é invocado. Este método não faz mais que verificar se a sessão já se encontrava num estado de *teardown* enviado pelo Session Manager. Caso a sessão já se encontre neste estado, nada é feito, caso contrário é enviada uma mensagem de *teardown* para o Session Manager. Quando o Session Manager recebe a mensagem de *teardown* faz todo o seu processamento e depois envia uma mensagem de *teardown completed* que vai entrar na fila de recepção de mensagens de comando. Quando o **InoSmil** faz **popIn()** deste comando, procura a sessão a que se destina, faz **stop** do **SMIL_player**, invoca o seu destrutor e de seguida processa todas as estatísticas, actualizando o número de sessões simultâneas, o número de sessões servidas, entre outros parâmetros.

5.4 Funcionalidades

Depois de implementado este interpretador, as seguintes funcionalidades foram conseguidas:

1. Leitura da configuração a partir de um ficheiro;
2. Cálculo da duração de cada sessão antes da sua execução: Quando é recebido um pedido de descrição de uma sessão, a sua duração é calculada percorrendo a árvore do documento uma vez, se este não precisar de pedir a duração de nenhum recurso multimédia, ou duas vezes se o pedido desta duração for necessário;
3. Vídeos em RTSP: Quando é percorrida a árvore do documento pela primeira vez é também verificado se existe algum vídeo em RTSP e quando é encontrado é guardado o seu endereço para posterior envio ao Session Manager aquando do envio da duração da sessão.
4. Texto: O interpretador suporta a reprodução de textos, desde que estes se encontrem definidos num ficheiro, através do atributo **src**, tal como o Ambulant Player suporta.

5. Elementos de sequência: É possível utilizar este tipo de elementos, que através do uso dos atributos **repeatCount** e **repeatDur** permitem a implementação de *playlists.*
6. Elementos paralelos: É possível utilizar elementos paralelos que permitem a implementação de funcionalidades de *text-overlay, logo-insertion* e *picture-in-picture.*
7. Uso de regiões: Quando o documento SMIL é percorrido pela primeira vez para o cálculo da duração da sessão, as regiões definidas no cabeçalho do documento são guardadas, com as suas propriedades, para serem passadas ao Session Manager aquando da reprodução dos elementos multimédia, se estes contiverem uma região com propriedades específicas definidas.
8. *Picture-in-picture*: Derivado do suporte para o uso de elementos paralelos e de regiões a funcionalidade de *picture-in-picture* é suportada pelo interpretador.
9. Inserção de logótipos: Derivado do suporte para elementos paralelos e de regiões a funcionalidade de inserção de logótipos é suportada pelo interpretador.
10. *Text-overlay*: Através do suporte para elementos do tipo paralelo e do suporte para texto, o interpretador implementa a funcionalidade de *text-overlay;*
11. *Logging*: Através do uso da biblioteca da Portugal Telecom Inovação, o interpretador guarda informações relativas ao seu funcionamento em ficheiro definido através de um ficheiro de configuração e o seu nível de detalhe é definido no mesmo ficheiro.
12. Portabilidade e Escalabilidade: Uma vez que o endereço do porto de comunicação com o Session Manager encontra-se definido num ficheiro de configuração e derivado de o interpretador ter um identificador próprio é possível ter o interpretador em máquinas diferentes das do restante sistema e várias instâncias do interpretador a funcionar em simultâneo.

6. Avaliação e Testes

Apesar do facto da biblioteca do Ambulant Player não ter sido implementada prevendo a sua utilização num ambiente *multi-thread*, com múltiplas instâncias do seu *player* a correr em simultâneo, os resultados dos testes não comprometem a sua utilização para a construção do interpretador. A prova de conceito mostrou ser possível a sua utilização neste tipo de ambiente e para esta arquitectura, apesar das fugas de memória detectadas.

Durante a compilação da libAmbulant verificou-se que afinal as fugas de memória eram provocadas pela falta da definição de algumas *flags* na *makefile* que compilava o projecto, levando a que na invocação de certos métodos, em especial nos destrutores, houvesse um desalinhamento no código.

Relativamente às funcionalidades, estas foram repetidamente testadas com sucesso, com excepção do uso do atributo **repeatCount** em elementos do tipo paralelo e sequência. Existe um erro reportado na página do projecto do Ambulant Player quanto ao uso deste atributo nos ditos elementos. Sempre que este é usado com um valor finito e maior que um nestes elementos, estes são repetidos esse número de vezes contudo, na última iteração, os elementos são reproduzidos continuamente, não parando após a duração definida. A equipa de desenvolvimento do Ambulant Player já tomou conta do caso e espera corrigir o problema na próxima versão do *player*, com data de lançamento prevista para Dezembro de 2009. Corrigido este erro, basta apenas fazer a actualização da libAmbulant para que este problema também seja corrigido no CSMIL.

Apesar de ter sido elaborado um estudo sobre a utilização da libAmbulant em Windows, a sua implementação neste sistema operativo não foi realizada, uma vez que o WSS apenas tem uma versão Linux, não havendo necessidade desta implementação nem havendo maneira de testar o CSMIL posteriormente. No entanto, a realização da prova de conceito neste ambiente deixa em aberto esta possibilidade no futuro, sendo que o único entrave são os problemas de fuga de memória. Como a causa destes é a ausência de *flags* na compilação a solução destes problemas parece simples e não se prevê grandes dificuldades na adaptação do módulo.

Aprovado o módulo nos testes de funcionalidade, restava apenas realizar os testes de carga a este. O WSS tem dois modos de funcionamento distintos. O primeiro assenta no uso de hardware para fazer o *transcoding* da apresentação e o segundo recorre a uma *mediabridge* para o envio da apresentação, não havendo a possibilidade do uso de funcionalidades avançadas como o *picture-in-picture.* Em ambos os modos de funcionamento o CSMIL comporta-se exactamente da mesma maneira, apesar do restante sistema ter comportamentos e limitações diferentes, em especial no uso de hardware pois o número de sessões concorrentes depende da capacidade deste.

Para a realização dos testes de carga foi usado o ambiente Linux descrito na secção 5.2.3. Foi também usado sempre o mesmo documento SMIL, uma vez as funcionalidades já haviam sido testadas anteriormente. O documento SMIL usado encontra-se descrito na figura 29.

```
<?xml version="1.0">
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.1//EN"
            http://www.w3c.org/2005/SMIL21/SMIL21.dtd>
<smil>
    <head>
        <layout>
            <root-layout id="Geral" width="300" height="300"/>
        </layout>
    </head>
    <body>
        <seq>
            <video src="magia.3gp" region="Geral" dur="20s"/>
        </seq>
    </body>
</smil>
```

Figura 29 - Documento SMIL usado nos testes de carga

Tal como é possível ver na figura 29, este documento reproduzia apenas um vídeo durante 20 segundos. Para auxiliar na realização destes testes foi usado também um *script* que utilizava o openRTSP[19] para realizar pedidos de novas sessões. Neste script era possível definir a duração de cada sessão, o número de sessões simultâneas e o intervalo para a criação de cada nova sessão. Deste modo, cada teste era previamente preparado e definida a carga que iria ser aplicada ao módulo. Com estes testes é testado todo o sistema e não apenas o CSMIL isoladamente. A tabela 15 mostra os resultados obtidos numa primeira série de testes realizados.

Ensaio	Número de Sessões Servidas	Número Máximo de Sessões Simultâneas	Número Máximo de Novas Sessões por Segundo
1	11281	99	14
2	16925	181	39
3	21960	180	33

Tabela 15 - Resultados obtidos na primeira série de testes de carga

O primeiro ensaio deste teste pretendia fazer um teste de robustez ao CSMIL, tentando criar um número reduzido de novas sessões por segundo e um número de sessões concorrentes relativamente longe do valor pretendido para ver se o sistema aguentava. Este teste falhou com um *segmentation fault* do CSMIL e os *logs* foram analisados em busca do motivo da falha. Estes não foram muito claros, uma vez que não apareciam erros e todas as sessões pareciam ter sido realizadas de forma normal, por isso foram realizados mais dois ensaios com uma carga mais elevada para se atingir este ponto rapidamente para análise dos acontecimentos. Os dois ensaios seguintes falharam da mesma forma que o primeiro e os *logs* do WSS foram analisados. Após a análise destes mesmos, constatou-se que em cada série de novas sessões criadas apenas uma sessão era terminada pelo CSMIL, sendo as restantes terminadas a pedido do Session Manager. Assim, foi revista a parte do CSMIL que processava os pedidos de término de sessão pelo Session Manager e foi descoberto que o destrutor do *player* da libAmbulant não estava a ser invocado neste caso, havendo uma fuga de memória que originava o erro. Descoberta a causa do problema uma nova série de testes foram realizados. Nesta segunda série de testes foi usada a carga máxima suportada pelo hardware, ou seja, cento e oitenta sessões concorrentes e cerca de trinta e cinco pedidos de novas sessões por segundo. A tabela 16 apresenta os resultados obtidos.

Ensaio	Número de Sessões Servidas	Número Máximo de Sessões Simultâneas	Número Máximo de Novas Sessões por Segundo
1	23201	180	34
2	21067	180	34
3	23498	180	34
4	22430	180	34
5	21628	180	34

Tabela 16 - Resultados obtidos na segunda série de testes de carga

Todos os cinco testes realizados nesta série apresentaram o mesmo problema. De acordo com os *logs* do CSMIL, era recebido um pedido de descrição de uma sessão, a resposta era dada e de seguida era recebido um pedido de término de sessão do Session Manager, verificando-se este problema durante tempo indefinido, no qual nenhuma nova sessão era iniciada. O Session Manager tem um parâmetro no qual se define o tempo de resposta a um pedido de nova sessão. Esse valor encontra-se definido a um segundo como padrão e uma análise atenta dos *logs* mostrou que este período de tempo era ultrapassado. Após a formulação de várias hipóteses para o problema passou-se a guardar no *log* do CSMIL o tamanho da fila de recepção de mensagens para despiste dos acontecimentos no intervalo de tempo com início na recepção do pedido até ao momento do envio da resposta. O *script* que usa o openRTSP tem a particularidade de que sempre que falha o início de uma nova sessão instantaneamente voltar a fazer outro pedido até que esta seja iniciada, tentando manter sempre o número de sessões concorrentes definido. Após a análise dos *logs* detectou-se que a certo ponto o tamanho da fila ia aumentando até cerca de mil mensagens pendentes. Concluiu-se que era necessário proteger com um mutex o período entre analisar a primeira mensagem da fila e removê-la porque por vezes o ciclo principal da aplicação retirava a mesma mensagem várias vezes por esta não ter sido removida ainda. Deste modo, foi realizada uma terceira série de testes que se encontram descritos na tabela 17.

Ensaio	Número de Sessões Servidas	Número Máximo de Sessões Simultâneas	Número Máximo de Novas Sessões por Segundo
1	54409	180	34
2	56982	180	34
3	38188	180	34
4	21078	180	34
5	34981	180	34
6	45017	180	34
7	59672	180	34
8	61206	180	34
9	59752	180	34
10	58923	180	34
11	65812	180	34

Tabela 17 - Resultados obtidos na terceira série de testes de carga

Esta terceira série de testes foi marcada por sucessivas interrupções em cada teste por diversos motivos. O primeiro ensaio desta série foi marcado pela falta de espaço em disco provocado pelos ficheiros de *log*. A acumulação de sucessivos ficheiros de *log* encheu o disco da máquina de testes levando a que houvesse um comportamento estranho por parte dos diferentes módulos, que levou a que o teste fosse interrompido. O segundo ensaio foi parado porque houve uma falha no *hardware* que levou a que as sessões não fossem reproduzidas, levando ao seu término. Assim o teste foi interrompido até que o problema fosse resolvido. Havendo apenas duas placas de *transcoding* disponíveis, com a falha da primeira apenas uma ficou disponível para os testes, ficando estes condicionados pelas necessidades da equipa. O terceiro, quarto, quinto e sexto ensaio foram marcados por interrupções sucessivas para ajustes na configuração do sistema pelo que nada se pode concluir destes. Os restantes ensaios foram marcados por interrupções devido à necessidade de utilização do sistema e da placa para testes no sistema já em produção na PT Inovação e no ensaio final com novos problemas na segunda placa que levou à interrupção dos testes.

Por motivos de agenda não foi possível a realização de novos testes sem a utilização da placa, no entanto uma última análise ao código foi feita utilizando uma ferramenta de depuração de código, o klocwork[10]. Esta ferramenta não encontrou *deadlocks*, fugas de memória nem outros erros como *null pointers* no código do CSMIL pelo que existem boas perspectivas quanto aos resultados de novos testes.

7. Conclusão e Trabalho Futuro

Com o aparecimento de novas tecnologias são produzidos dispositivos móveis mais rápidos, mais pequenos e com maiores capacidades de processamento. Isto leva a que haja capacidade para a implementação de novas funcionalidades, nomeadamente na área do mobile TV e vídeo-conferência. A linguagem SMIL mostrou ser uma excelente forma de coordenar e coreografar apresentações multimédia, permitindo implementar as funcionalidades de *picture-in-picture*, *text-overlay* e *logo-insertion*. Para o objectivo que se pretendia esta linguagem mostrou ser a adequada, permitindo o controle da temporização da reprodução de elementos multimédia e um controlo espacial do local de reprodução destes, sendo este um ponto fulcral na implementação das funcionalidades citadas anteriormente.

Apesar de vários *players* começarem gradualmente a aceitar as funcionalidades do SMIL, estas ainda não se encontram disponíveis para dispositivos móveis, surgindo assim esta ferramenta para o WSS, que pretende passar a parte de processamento dos conteúdos das apresentações para o lado do servidor, enviando para os clientes apenas uma *stream* com o resultado[20]. Desta forma, o CSMIL apresenta-se no contexto do WSS como um módulo de controlo da apresentação de conteúdos, podendo este ser opcional pois o sistema permite a reprodução de vídeos sem funcionalidades avançadas, ou seja, sem o CSMIL.

No que toca à concepção do interpretador, o Ambulant Player mostrou-se ser uma excelente escolha, permitindo que o interpretador reconhecesse a norma 3.0 do SMIL. Adicionalmente, sempre que a norma seja actualizada basta apenas actualizar a libAmbulant que o interpretador reconhecerá automaticamente a nova norma. O uso deste *player* reduziu significativamente o tempo de desenvolvimento, permitindo que o projecto se focasse em aspectos de robustez e eficiência. O sistema de *factories* do Ambulant Player simplificou bastante a tarefa de construção do interpretador uma vez que bastou apenas criar algumas *factories* para a sua transformação. No entanto, a parte de criação de novas *factories* não se encontra devidamente documentada, pelo que esta adaptação exigiu um estudo aprofundado deste. A adaptação deste *player* a diferentes contextos não se encontra documentada,

com excepção da alteração da sua interface gráfica, pelo que se pode concluir que este projecto foi pioneiro na utilização da sua biblioteca para outros fins que não a reprodução integral de documentos SMIL.

A execução de várias instâncias do Ambulant Player não apresentou problemas neste contexto podendo-se considerar um sucesso. Mais uma vez, e através de contactos efectuados através da *mailing list* do *player*, este tipo de aplicação nunca tinha sido efectuada com a biblioteca.

Por questões de agenda não foi possível concluir todos os testes desejados ao interpretador, no entanto, a nível de funcionalidades e carga este apresentou valores muito aceitáveis e que não afectam a sua utilização no WSS, em especial com o uso de funcionalidades avançadas. Fica por testar a robustez deste, no entanto, após a análise do código com o klocwork, espera-se que estes sejam efectuados com sucesso. A libAmbulant apresentou-se capaz para se atingir os objectivos propostos de eficiência e robustez.

Quanto a trabalho futuro, uma funcionalidade que seria interessante de implementar seria a possibilidade do uso de comandos de *pause* e *resume*, permitindo que o utilizador parasse a sessão e a retomasse posteriormente. Neste caso, o interpretador encontra-se preparado para suportar esta funcionalidade bastando apenas processar este novo tipo de mensagens. Esta funcionalidade não foi suportada no momento uma vez que o WSS não a suporta, no entanto fica em aberto para novas versões do sistema. Fica também para trabalho futuro a continuação da realização de testes de robustez ao interpretador, impossibilitados no momento por falhas de natureza técnica e questões de agenda.

8. Bibliografia

1. W3C, *Synchronized Multimedia Integration Language (SMIL 3.0),* Dezembro 2008;
2. Dick C. A. Bulterman, A. J. Jansen and Pablo Cesar, *Video on the web: Experiences from SMIL and Ambulant Annotator;*
3. World Wide Web Consortium, http://www.w3.org, 2009;
4. Ambulant Player, http://www.ambulantplayer.org, 2009;
5. Helix Community, Helix Player, http://player.helixcommunity.org, 2009;
6. SunTREC, Schmunzel, http://suntrec.salzburgresearch.at/schmunzel/, 2009;
7. X-Smiles, http://www.xsmiles.org, 2009;
8. Oratrix, GRiNS, http://www.oratrix.com/Products/G2P, 2008;
9. Apple, QuickTime, http://www.apple.com/quicktime/, 2009;
10. Klocwork Inc, Klocwork, http://www.klocwork.com/, 2009;
11. Expat, http://expat.sourceforge.net/, 2009;
12. Apache XML Project, Xerces-C++, http://xerces.apache.org/xerces-c/, 2009;
13. FFmpeg, http://ffmpeg.org/, 2009;
14. SDL, http://www.libsdl.org/, 2009;
15. Live Networks Inc, Live555, http://live555.com/, 2009;
16. GStreamer, http://gstreamer.freedesktop.org/, 2009;
17. The GTK+ Team, GTK, http://www.gtk.org/, 2009;
18. Lightweight Communications and Marshalling, http://code.google.com/p/lcm/, 2009;
19. Live Networks Inc, openRTSP, http://www.live555.com/openRTSP/, 2009;
20. Tayeb Lemlouma, Nabil Layiada, *SMIL Content Adaptation for Embedded Devices,* 2003;
21. Valgrind, http://valgrind.org/, 2009;
22. rsyslog, http://www.rsyslog.com/, 2009;