Departamento de Engenharia Informática

Interface HL7 para observation patterns

Sílvio Filipe Lopes Gonçalves

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Área de especialização em Arquitectura, Sistemas e Redes

Orientador
Prof. Doutor Ângelo Martins
Co-orientador
Artur Rocha

Júri

Presidente: Prof. Doutora Maria de Fátima Coutinho Rodrigues, Professora Coordenadora no Departamento de Engenharia Informática do Instituto Superior de Engenharia do Porto
Vogais: Prof. Doutor Lino Manuel Baptista Figueiredo, Professor Adjunto no Departamento de Engenharia Electrotécnica do Instituto Superior de Engenharia do Porto
Prof. Doutor Ângelo Manuel Rego e Silva Martins, Professor Adjunto no Departamento de Engenharia Informática do Instituto Superior de Engenharia do Porto

Porto, Outubro 2010

…aos meus Pais e ao meu Tio Curioso

Agradecimentos

O meu maior agradecimento é para os meus Pais, que acreditam nas minhas capacidades e sempre apoiaram as minhas decisões.

Aos meus irmãos e restante família, agradeço os momentos felizes que me proporcionam e me enchem de força para encarar os desafios.

Aos meus amigos, agradeço a amizade e o apoio que sempre me deram. Um agradecimento muito especial aos meus amigos, Fernando Piedade e Olga Oliveira, que ajudam na revisão dos textos.

Ao Professor Doutor Ângelo Martins, orientador da dissertação, agradeço o apoio, a partilha do saber e as valiosas contribuições para o trabalho.

Resumo

Com a generalização das tecnologias de informação e comunicação na área da saúde, a monitorização remota de pacientes, a partir de dispositivos móveis, é uma realidade que contribui para uma melhor qualidade de vida dos pacientes. Mas, não são só os pacientes que ganham com a introdução destas tecnologias, também os profissionais de saúde e os hospitais retiram as suas vantagens. Os profissionais de saúde são munidos de uma ferramenta móvel de monitorização permanente de sinais vitais, aumentando desta forma a ligação médico-paciente. Relativamente aos hospitais, estes vêem os custos de manutenção reduzirem, em virtude da possibilidade dos pacientes puderem ser monitorizados a partir dos seus lares.

Neste projecto pretende-se identificar mecanismos que possibilitem responder de forma eficaz à necessidade de partilha de informação, nomeadamente protocolos de comunicação, segurança e sistemas de integração. Foi projectado um protótipo, constituído por um *middleware* e uma aplicação cliente móvel, onde o *middleware* tem como missão garantir a interoperabilidade entre um servidor HL7 e a aplicação cliente, utilizada pelo profissional de saúde. A normalização da informação médica trocada entre o servidor HL7 e o *middleware* obedece à norma internacional HL7.

Palavras-chave:
Sistemas de monitorização remoto, *protocol buffer*, *web service*, segurança, HL7, *middleware*

Abstract

With the spread of information and communication technologies in health care the remote monitoring of patients from mobile devices is a fact that contributes to improve patient's quality of life. But not only are the patients who win with the introduction of these technologies, also health professionals and hospitals derive their benefits. Health professionals are equipped with a mobile tool to continuously monitor vital signs, thus increasing the connection doctor-patient relationship. For hospitals, they reduce maintenance costs due to the possibility of patients being monitored from their homes.

This project aims to identify mechanisms that can respond effectively to the need for information sharing, including the communications protocols, security and integration systems. A prototype was designed, consisting of a middleware and a mobile client application where the middleware's mission is to ensure interoperability between an HL7 server and client application, used by health professionals. The standardization of medical information exchanged between the server and HL7 middleware follows the international standard HL7.

Keywords:
Remote monitoring systems, protocol buffer, web services, security, HL7, middleware

Índice

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

HL7 – Health Level Seven

GPRS – General Packet Radio Service

HTTP – Hypertext Transfer Protocol

SOAP – Simple Object Access Protocol

XML - Extensible Markup Language

J2ME – Java 2 Platform, Micro Edition

J2SE – Java 2 Platform, Standard Edition

API – Application Programming Interface

CLPC – Connected Limited Device Configurations

MIDP – Mobile Information Device Profile

TLS – Transport Layer Security

DES – Data Encryption Standard

AES – Advanced Encryption Standard

RSA – Rivest, Shamir e Adleman

MOM – Message Oriented Middleware

1. Introdução

As constantes deslocações a unidades de saúde para monitorização de sinais vitais ou outro tipo de métricas, provocam nos pacientes um desconforto, sendo que em certas situações essa monitorização poderá mesmo não ser realizadas por questões de limitações de ordem física, económica ou geográfica. Os sectores mais vulneráveis da sociedade, especialmente a população idosa, doentes crónicos e deficientes físicos, sentem com mais intensidade estas restrições. A introdução de sistemas de monitorização de pacientes, baseados em tecnologias de informação e comunicação, reduzem as limitações apontadas anteriormente, desempenhando desta forma um papel determinante na melhoria da qualidade de vida dos pacientes. Com a introdução de sistemas de saúde com estas características, a necessidade do paciente ganha maior predominância. O paciente poderá decidir se pretende ser monitorizado no hospital ou em sua casa, sem prejuízo na qualidade dos serviços médicos prestados.

As vantagens da implementação destes sistemas não são só para os pacientes, também os profissionais de saúde retiram as suas vantagens. A possibilidade do profissional de saúde aceder a dados de determinado paciente a partir de um vulgar dispositivo móvel, aumenta a proximidade entre as duas partes envolvidas, criando uma relação de confiança. Esta ligação permanente permite a detecção e aviso prévio de eventuais anomalias.

Esta realidade só é possível com a evolução que as tecnologias de informação e de comunicação têm registado. A combinação entre estas tecnologias e a prestação de serviços de saúde é uma área de grande interesse para as respectivas comunidades investigadoras, tendo sempre como objectivo final a melhoria da qualidade dos serviços médicos prestados e uma melhor eficiência na gestão dos recursos (humanos, logísticos e financeiros), colocando o paciente como o ponto central de todo o processo de prestação de cuidados de saúde.

1.1 Enquadramento

Esta dissertação surge no seguimento do projecto europeu **Caalyx,** onde foi desenvolvido um mecanismo de monitorização médica baseada no conceito das *observation patterns* (padrões de observação) da OGC (**Open Geospatial Consortium**). Este mecanismo de observações é distribuído e hierárquico, sendo cada observação responsável por efectuar localmente a análise do(s) parâmetro(s) que monitoriza.

Na sequência do projecto **Caalyx**, nasceu o projecto europeu **AAL eCaalyx** que pretende aprofundar o trabalho desenvolvido, designadamente na abordagem da ligação de sistemas de monitorização distribuídos a sistemas de informação com os registos médicos dos indivíduos monitorizados, tendo como base as normas *Health Level Seven* (HL7).

1.2 Objectivos

Na procura de melhorar os serviços de saúde prestados aos pacientes, a comunidade médica, em parceria com a comunidade ligada às novas tecnologias, têm mobilizado esforços no sentido de conceber novas soluções que apresentem vantagens não só para os pacientes, mas também para as várias entidades envolvidas em todo o processo de prestação de serviços de saúde.

Nesta dissertação, existe um conjunto de objectivos para os quais foram realizados estudos e testes no sentido de avaliar diferentes formas de solucionar determinados problemas. A seguir, são enumerados os principais objectivos do projecto.

Protocolos de comunicação

A forma como a informação é estruturada e transmitida entre as várias entidades é um factor importante para o tempo de resposta do sistema. O estudo de um protocolo de comunicação ideal para o projecto é um dos objectivos. Neste sentido, foram analisados e testados dois protocolos de comunicação, o *protocol buffer* e os *web services.*

Segurança

A partilha de informação é uma característica subjacente a este tipo de soluções móveis. A informação partilhada é extremamente sensível, devendo ter acesso à mesma, os seus titulares e as várias entidades legalmente habilitadas para o efeito, nomeadamente profissionais de saúde e unidades de saúde. Neste sentido, torna-se necessário realizar estudos, tendo como objectivo a definição de um mecanismo de segurança capaz de conferir o nível de segurança exigido para a informação partilhada.

Middleware

Actualmente, existe uma grande diversidade de arquitecturas que resulta, por vezes, na existência de algumas incompatibilidades entre sistemas. O *middleware* é a camada intermédia entre a aplicação cliente e o servidor de registos clínicos que resolve eventuais problemas de incompatibilidades, garantido interoperabilidade entre arquitecturas com características diferentes. Um dos objectivos deste projecto é desenvolver um *middleware*, que para além de implementar o conceito de interoperabilidade, apresenta o grau de fiabilidade desejado.

A combinação dos vários objectivos enumerados anteriormente, resulta na obtenção do objectivo principal: desenvolver um sistema móvel de acesso a registos clínicos, previamente monitorizados e transferidos para um ponto central, utilizando como camada intermédia um sistema *middleware.*

1.3 Estruturação

Esta dissertação está estruturada em seis capítulos. A seguir, são apresentados e descritos, de forma sucinta, os assuntos abordados nos diferentes capítulos.

O Capítulo 1 é introdutório e apresenta uma visão geral do assunto a tratar. No Capítulo 2 é descrito um estado da arte geral, ou seja, são apresentados vários conceitos relacionados com soluções de saúde móvel. Este estado da arte é geral porque para cada um dos objectivos definidos anteriormente, protocolos de comunicação, segurança e *middleware*, é apresentado um estado da arte mais pormenorizado, a definir no respectivo capítulo. Este capítulo também contém uma visão geral do projecto em estudo. No Capítulo 3 são analisados e implementados dois protocolos de comunicação diferentes, o *protocol buffer* e os *web services*. O Capítulo 4 é dedicado à segurança, sendo apresentadas noções gerais de segurança que serviram de base à definição de um modelo que visa criar um ambiente seguro para a partilha de informação. O Capítulo 5 define as estruturas de mensagens partilhadas entre as várias entidades e introduz a norma HL7. No Capítulo 6 são introduzidos conceitos relacionados com a implementação de soluções *middleware,* baseadas em padrões. A partir desta análise, projectou-se uma solução que preenche os requisitos pretendidos para o projecto em estudo. No Capítulo 7 é apresentada a conclusão final do projecto e os respectivos trabalhos futuros.

2. Estado da Arte

A evolução da medicina aponta para o reforço da componente preventiva, especialmente ao nível dos pacientes idosos e de risco. Neste sentido, a monitorização remota e automática de sinais vitais, como o batimento cardíaco, tensão arterial, temperatura do corpo, entre outros, é um aspecto importante a considerar. A introdução de novas ferramentas de monitorização contribui para o reconhecimento de eventuais anomalias, sem condicionar o estilo de vida da pessoa monitorizada. A monitorização continuada de sinais vitais introduz uma nova visão de avaliação de pacientes, transformando a monitorização num processo individual, baseado no comportamento e histórico clínico de cada paciente (Crk *et al.*, 2009).

O envelhecimento da população é uma realidade para a qual as sociedades actuais têm obrigação de encontrar uma solução, sendo este tipo de público, caracterizado por apresentar dificuldades de locomoção, um alvo onde estas tecnologias proporcionam grandes benefícios. Prevê-se que os profissionais de saúde não tenham capacidade de resposta para todas as solicitações, existindo, por isso, a necessidade de projectar soluções viáveis, como a monitorização remota, que vão de encontro às necessidades dos pacientes e estejam ao alcance dos recursos económicos de estados sobrecarregados financeiramente.

Actualmente, os sistemas de informação na área da saúde são sobretudo utilizados para troca e armazenamento de informação clínica. Ao contrário do processo de monitorização que é realizado presencialmente, as soluções de saúde móveis recolhem sinais vitais, avaliam e processam os dados e os enviam para um centro de alojamento de registos clínicos.

2.1 Vantagens das Soluções de Saúde Móveis

A introdução das tecnologias de informação e comunicação ao serviço da saúde traz inúmeros benefícios, nomeadamente para os pacientes, os profissionais de saúde e os hospitais.

Benefícios para os pacientes:

- Maior interacção entre profissional de saúde e paciente;
- Redução dos custos na prestação de cuidados de saúde;
- Pró-actividade e prevenção.

Benefícios para os profissionais de saúde

- Acesso e gestão de informação médica, independentemente da localização;
- Monitorização remota de pacientes;
- Coordenação na realização de tarefas médicas, como realização de exames médicos, ou ministração de medicamentos.

Benefícios para os hospitais

- Aumento da produtividade;
- Diminuição dos gastos com papel;
- Redução do tempo de comunicação de informação médica;
- Diminuição da taxa de ocupação.

2.2 Limitações das Soluções de Saúde Móveis

A utilização de soluções de saúde móveis apresenta ainda algumas debilidades que tendem a desaparecer com o tempo. Os dispositivos móveis utilizados neste tipo de soluções apresentam limitações de processamento, autonomia e memória (Gaddah *and* Kunz, 2003). A autonomia é possivelmente o principal obstáculo na evolução de soluções mais robustas. A gestão equilibrada da energia, ganha por isso, uma importância de relevo na concessão e utilização deste tipo de soluções.
O artigo de Crk *et al.* (2009) descreve estudos realizados que ajudam a compreender e a implementar soluções móveis com uma gestão ediciente da energia.

A implementação de soluções que pressupõem a existência permanente de uma ligação com o servidor central, enfrentam o problema da falta de cobertura de rede móvel. Apesar da tendência ser para diminuir, existem ainda zonas com fraca cobertura de rede móvel. Estas redes, ao contrário das redes de dados tradicionais (por cabo), são ainda caracterizadas por limitações de largura de banda, falhas de conexão e erros de transmissão (Gaddah *and* Kunz, 2003).

2.3 Telemedicina

A telemedicina refere-se ao uso de informação electrónica e tecnologias de comunicação, que providenciam e suportam a prestação de cuidados de saúde quando os participantes (profissional de saúde e paciente) estão geograficamente separados (Deng *and* Poole, 2003). O recurso a esta tecnologia proporciona um acompanhamento permanente da evolução clínica do paciente por parte dos profissionais de saúde, sem que haja qualquer tipo de deslocamento físico.

Num mundo cada vez mais globalizado, a centralização e partilha de informação clínica é um factor que ganha grande importância. A telemedicina desempenha um papel relevante na estruturação

e manutenção de uma rede de informação médica. As vantagens, não são só no acesso a dados clínicos de pacientes, mas também no conhecimento e aprendizagem que os profissionais de saúde podem adquirir e partilhar.

Nos últimos anos este conceito sofreu grandes modificações, nomeadamente na implementação prática. O factor que mais contribuiu para essa evolução está relacionado com os avanços tecnológicos observados nos últimos tempos, nomeadamente ao nível da comunicação.

A telemedicina não é apenas inovação tecnológica, mas também sócio-cultural. Se por um lado, a telemedicina é um sistema tecnológico adoptado como meio de comunicação para a transferência de informação médica, ou para a partilha da mesma entre instituições de saúde; por outro lado, é sistema sócio-organizacional inovador que tem implicações na prestação estruturada e organizada de cuidados de saúde (Deng *and* Poole, 2003).

2.4 Projectos de Sistemas de Saúde Móveis

A evolução das tecnologias de informação e de comunicação permitiu aumentar o campo de investigação nesta área, possibilitando a criação de sistemas revolucionários que visam proporcionar melhor qualidade na prestação de cuidados de saúde. Existem inúmeros projectos nesta área onde a inclusão de tecnologias de informação e de comunicação, nomeadamente dispositivos móveis, é uma característica comum à grande maioria. A seguida, são descritos, de forma sucinta, alguns projectos que abordam a temática em estudo nesta dissertação.

2.4.1 Sistema de Saúde baseado em Sistema de Multi-Agentes

O artigo de Chan *et al.* (2008) descreve a arquitectura de um sistema de monitorização baseada em múltiplos agentes, onde cada um representa o papel dos diferentes intervenientes num determinado processo clínico (médico, pacientes, enfermeiro, …). Os agentes são implementados em *Java* e executam em dispositivos móveis (*PDA*), sendo incorporados em três áreas: dispositivo móvel do paciente, dispositivo móvel do profissional de saúde e outros dispositivos móveis (*notebook*) e servidores.

Estes agentes não restringem o seu funcionalmente à simples recolha e envio de informação clínica, são também agentes preventivos que têm a capacidade de detectar e reportar eventuais anomalias que possam surgir.

A arquitectura de rede é constituída por clientes (dispositivos móveis) e servidores que providenciam comunicação e coordenação, dispostos em três níveis de redes, *body area network* (*BAN*), *personal area network* (*PAN*) e *wide area network* (*WAN*). A rede *BAN* é utilizada na comunicação entre o dispositivo responsável pela recolha de vários tipos de parâmetros vitais e o

dispositivo móvel responsável pelo processamento dos dados. A tecnologia de comunicação utilizada para a transmissão de dados, entre dispositivo, é o *Bluetooth*. A rede *PAN* possibilita a transmissão de dados recolhidos a partir da *BAN* com entidades externas (médico, hospital, …). O agente responsável pela transferência dos dados estabelece comunicação com o servidor recorrendo à tecnologia *GPRS* ou *3G*. A informação é cifrada e enviada no formato de mensagem para o servidor via *HTTP* ou via *web services*. A rede *WAN* providencia conectividade entre as várias partes envolvidas em todo o processo clínico. A Figura 1 ilustra o funcionamento da arquitectura do sistema de saúde baseado em sistema multi-agentes.

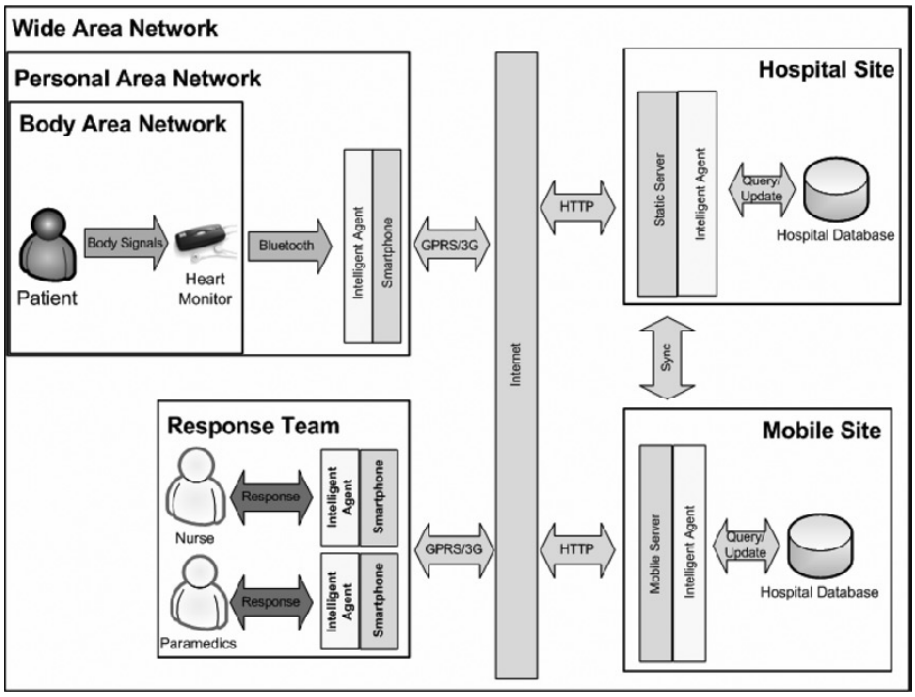

Figura 1 – Arquitectura do sistema de saúde baseado em agentes (Chan *et al.*, 2008)

2.4.2 Aplicação Médica Móvel baseada em *Web Services*

A capacidade de transmissão de informação é uma característica presente em vários sistemas de saúde móveis. Existem vários estudos que abordam diversas formas de implementação e transmissão de informação. O estudo descrito no artigo de Adnan *and* Hashim (2009), visa demonstrar o funcionamento de uma aplicação médica móvel baseada em *web services*. Os autores desenvolveram duas aplicações, uma aplicação móvel que executa num dispositivo móvel e outra que funciona como camada intermédia entre a aplicação móvel e a base de dados desenvolvida em *MySQL*. As aplicações, cliente e servidor, foram implementadas na tecnologia *J2ME* e *J2EE*, respectivamente. A Figura 2 mostra o modo de funcionamento da arquitectura descrita.

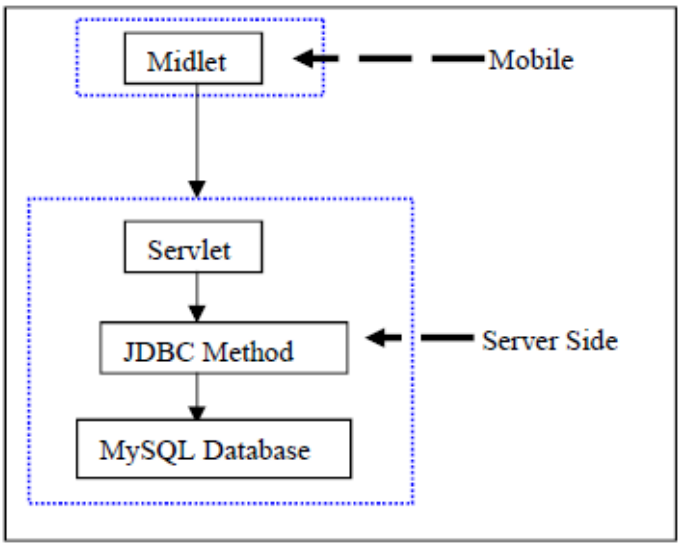

Figura 2 – Arquitectura de uma aplicação médica móvel baseada em *web services* (Adnan and Hashim., 2009)

2.5 Sistemas de Saúde Móveis Comerciais

2.5.1 MobiMed

MobilMed (2006) é uma empresa especializada na implementação de aplicações de saúde para hospitais. O principal produto da MobiMed consiste numa plataforma, designada *MMF* (MobiMed *framework*), a partir da qual é possível desenvolver aplicações de saúde móveis. As aplicações desenvolvidas permitem assistir os profissionais de saúde no desempenho das suas funções e executam em dispositivos móveis, como *Pocket PC* e *Tablet PC*. Estas aplicações apresentam uma interface intuitiva e são utilizadas na monitorização e transmissão de sinais vitais de pacientes para os profissionais de saúde que se encontram, por exemplo num hospital. Actualmente, a MobiMed disponibiliza três soluções móveis de saúde:

- **MobiDoc** – Possibilita a comunicação e actualização de dados clínicos de pacientes na base de dados central;
- **MobiNurse** – Assiste o enfermeiro na execução do plano de tarefas, possibilitando ainda a divisão de tarefas pelos profissionais de saúde;
- **MobiFood** – Permite gerar a dieta apropriada para determinado paciente e personaliza essa mesma dieta via dispositivo móvel.

Todas as aplicações são caracterizadas pela sua versatilidade, ou seja, para além de puderem ser executadas em dispositivos móveis, podem também ser executadas em computadores pessoais.

2.5.2 Mobile-CliniQ™

Moblie-CliniQ™ (Aerotel, 2010) é uma aplicação que executa em telemóveis com sistema operativo *Symbian*. A aplicação permite medir vários parâmetros médicos e comunicar, através da interface *Bluetooth*, com dispositivos de monitorização de sinais vitais. As medições recolhidas são enviadas via *GPRS* para um servidor central. A finalidade do desenvolvimento do Mobile-CliniQ™ é proporcionar aos utilizadores um conjunto de serviços médicos, para os quais não é necessário a intervenção de um profissional de saúde. Os benefícios práticos da utilização da aplicação são, a transmissão *wireless* dos dados monitorizados, o aumento da interacção médico-paciente e a recepção automática no *call center*. A Figura 3 mostra o modo de funcionamento da aplicação.

Figura 3 – Arquitectura do sistema Mobile-CliniQ™ (Aerotel, 2010)

2.6 Projecto eCaalyx

Esta dissertação surge no seguimento das várias soluções móveis de prestação de serviços médicos apresentadas. As principais diferenças residem na abordagem tecnológica do problema. No entanto, o objectivo final continua a ser o mesmo, melhorar os cuidados de saúde prestados e a qualidade de vida dos pacientes.

Este estudo insere-se no projecto europeu **AAL eCaalyx**[1] (2009-2012) cujo objectivo principal é desenvolver uma arquitectura capaz de proporcionar a troca de informação médica entre sistemas de monitorização distribuídos (dispositivos móveis) e um sistema centralizado de registos médicos, em que a troca de informação obedece à norma HL7. A Figura 4 mostra a visão geral do projecto eCaalyx.

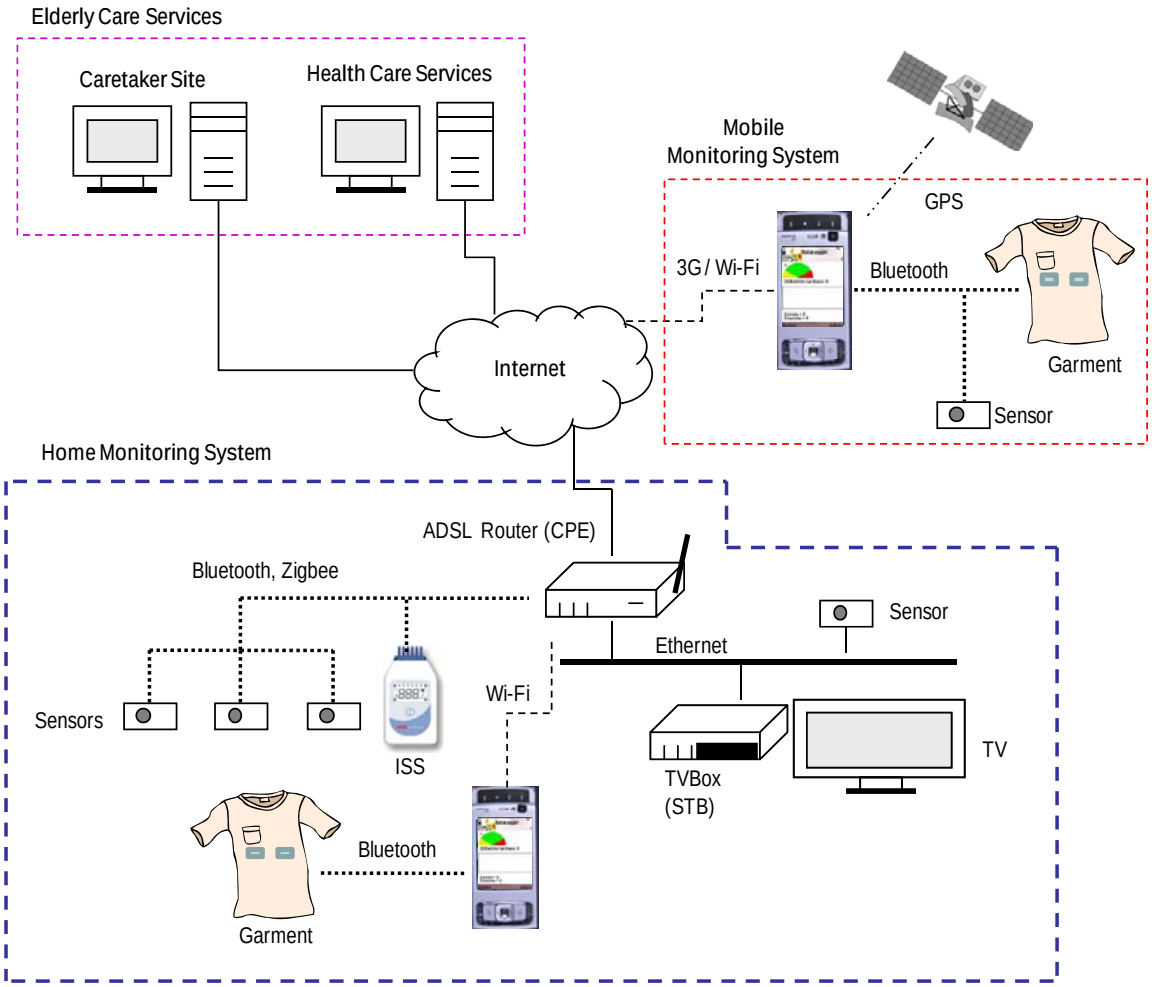

Figura 4 - Visão do projecto eCaalyx

[1] http://ecaalyx.org

O processo de implementação de sistemas distribuídos com estas características exige um estudo exaustivo, visando ultrapassar as limitações apresentadas pelas várias tecnologias subjacentes a todo o sistema. Neste sentido, o sistema proposto foi dividido em vários pontos que são a seguir apresentados.

Aplicação Cliente

O sucesso de qualquer aplicação depende em grande parte da aplicação cliente. O seu papel é determinante, uma vez que é o espelho de todo o sistema, no entanto, o processo que envolve o desenvolvimento de aplicações que executam em ambientes móveis é complexo e demoroso. As dificuldades advêm das limitações impostas pelos recursos (hardware) dos dispositivos, pelo tamanho e resolução dos visores e pelo facto das ferramentas subjacentes ao processo de desenvolvimento deste tipo de soluções serem em número reduzido e limitado.

Relativamente à usabilidade da aplicação, esta deve ser funcional, simples, intuitiva e fácil de manusear.

Protocolos de Comunicação

O acesso em tempo útil à informação, nomeadamente em sistemas de saúde, é um factor de extrema importância já que neste caso especifico pode salvar vidas. A massificação das redes móveis levou ao surgimento de soluções na área da saúde. Estes sistemas visam proporcionar uma monitorização de proximidade, ou seja, capacidade de profissionais de saúde terem acesso a dados clínicos em tempo real independentemente da sua localização. O tempo que decorre desde o pedido até à obtenção da resposta e o custo de comunicação são aspectos relevantes para o sistema em estudo. Neste sentido, procedeu-se à análise e implementação de dois protocolos de comunicação, o *protocol buffer* e os *web services*.

Segurança

A segurança é um requisito essencial em sistemas de saúde. A circulação de informação clínica, classificada como confidencial, através da rede, pode implicar a consulta da mesma por indivíduos não autorizados. A implementação de mecanismos de protecção é por isso um aspecto indispensável (Boukerche *and* Ren, 2009). As limitações impostas pelos dispositivos móveis inviabilizam o desenvolvimento de mecanismos de segurança complexos, sendo necessário ajustar aos recursos disponíveis os critérios de segurança exigidos. O mecanismo de segurança tem de ser simples e funcional, sem, no entanto, representar um custo muito penoso para o objectivo final de todo o sistema, ou seja, aceder em tempo útil aos dados solicitados.

Middleware

O *Middleware* ou camada intermédia, é um componente desenhado para ajudar na gestão da complexidade e heterogeneidade inerentes a sistemas distribuídos. É uma camada de software entre o sistema (sistema operativo ou outro tipo de sistema) e a aplicação, fornecendo uma interface programável comum (*API*) a todo o sistema distribuído, tornando o processo de desenvolvimento de aplicações mais fácil. Krakowiasky define *middleware* da seguinte forma:

> *"In a distributed computing system, middleware is defined as the software layer that lies between the operating system and the applications on each site of the system. Its role is to make application development easier, by providing common programming abstractions, by masking the heterogeneity and the distribution of the underlying hardware and operating systems, and by hiding low-level programming details."* (Krakowiak, 2009)

A introdução de um sistema *middleware* apresenta inúmeras vantagens, nomeadamente:

- Encapsulamento da informação;
- Aumento da segurança;
- Interoperabilidade entre plataformas distintas;
- Interface programável (*API*).

Neste projecto, o *middleware* é responsável pelo tratamento de pedidos realizados por clientes. O tratamento consiste em gerir os pedidos e solicitar ao servidor HL7, através da utilização da norma HL7, resposta aos mesmos.

2.7 Tecnologia utilizada

O projecto foi implementado na linguagem *Java*. As aplicações *desktop* e móveis desenvolvidas utilizam a tecnologia *J2SE* e *J2ME*, respectivamente. A plataforma *J2ME* é composta, entre outros, por dois componentes fundamentais: *Connected Limited Device Configurations* (*CLDC*) e o *Mobile Information Device Profiles* (*MIDP*). *CLDC* é uma especificação de configuração mínima em termos de hardware e bibliotecas padrão para dispositivos móveis (Sun Microsystems, 2003a). Esta especificação apresenta actualmente duas versões, o *CLDC* versão 1.0 (JSR[2] 30) e o *CLDC* versão 1.1 (JSR 139), sendo esta última a utilizada neste projecto, pelo facto de incorporar

[2] JSR – Java Specification Requests

funcionalidades que a anterior versão não suportava, como por exemplo, a introdução das *classes Float* e *Double* (Sun Microsystems, 2003a). O *MIDP*, que executa sobre a camada *CLDC*, define um conjunto de bibliotecas que são mais específicas a uma determinada categoria de dispositivos que as bibliotecas disponibilizadas pelo *CLDC* (Sun Microsystems, 2003a). Actualmente, a versão mais recente do *MIDP* é a versão 3.0 (JSR 271).

A plataforma de desenvolvimento escolhida foi o *NetBeans*, por ser um ambiente de desenvolvimento integrado (*IDE*) de software gratuito e de código aberto para várias linguagens de programação, incluindo o *Java*. O *NetBeans* disponibiliza um conjunto de ferramentas necessárias para a criação de aplicações de *desktop*, empresariais, *Web* e móveis multi-plataformas (Wikipédia, 2010c).

3. Protocolos de Comunicação

A proliferação dos sistemas de informação permitiu criar uma relação de proximidade entre a informação e a sociedade, através da constituição de uma rede de dados, onde a partilha de informação é uma constante. Neste contexto, a informação representa um papel preponderante na tomada de decisões, independentemente da actividade praticada. A forma como a informação é processada e partilhada depende da tecnologia que lhe dá suporte, sendo, por isso, extremamente importante padronizar processos e mecanismos de transmissão de informação, assegurando a transparência no acesso.

Surgiram nesta área várias abordagens muito interessantes que através das suas características inovadoras conquistaram o mercado, sendo os *web services* um bom exemplo desse sucesso.

A empresa Google revelou recentemente um projecto *open source*, denominado *protocol buffer*, adequada à troca de mensagens curtas, que pretende reduzir algum do *overhead* dos *web services* (Google Code, 2010a).

Na área da saúde, as tecnologias de informação representam um papel determinante no acesso permanente à informação, nomeadamente, a registos clínicos. O recurso a estas tecnologias permite, não só aceder a registos clínicos, como proporcionar uma monitorização constante de pacientes, independente da localização geográfica.

No projecto em estudo, o acesso à informação é um factor determinante no sucesso do sistema de saúde a desenvolver. Para ajudar na escolha da tecnologia de transmissão de informação procedeu-se ao estudo de duas tecnologia que existem actualmente, *web services* e *protocol buffer*.

3.1 Web Services

A arquitectura *web services* é um padrão desenvolvido pela W3C baseada na troca de mensagens que prende resolver os problemas de interoperabilidade entre diferentes plataformas (W3C, 2004b). Esta arquitectura baseia-se em tecnologias como *HTTP*, *XML*, *SOAP*, entre outras.

O *HTTP* é um protocolo da camada aplicacional utilizado para estabelecer conexões entre sistemas de informação, que tem como principal objectivo possibilitar a transferência de informação, sob a forma de mensagem.

A comunicação de dados é feita utilizando a tecnologia *XML*, garantindo interoperabilidade entre ambientes diferentes.

SOAP é um protocolo simples e leve, que possibilita a troca de informação estruturada entre pontos que operam em ambientes descentralizados e distribuídos. *SOAP* baseia-se na tecnologia *XML*, sendo prática comum utilizar como protocolo de transporte o *HTTP* (W3C, 2004a).

A Figura 5 ilustra o funcionamento da arquitectura *web service*.

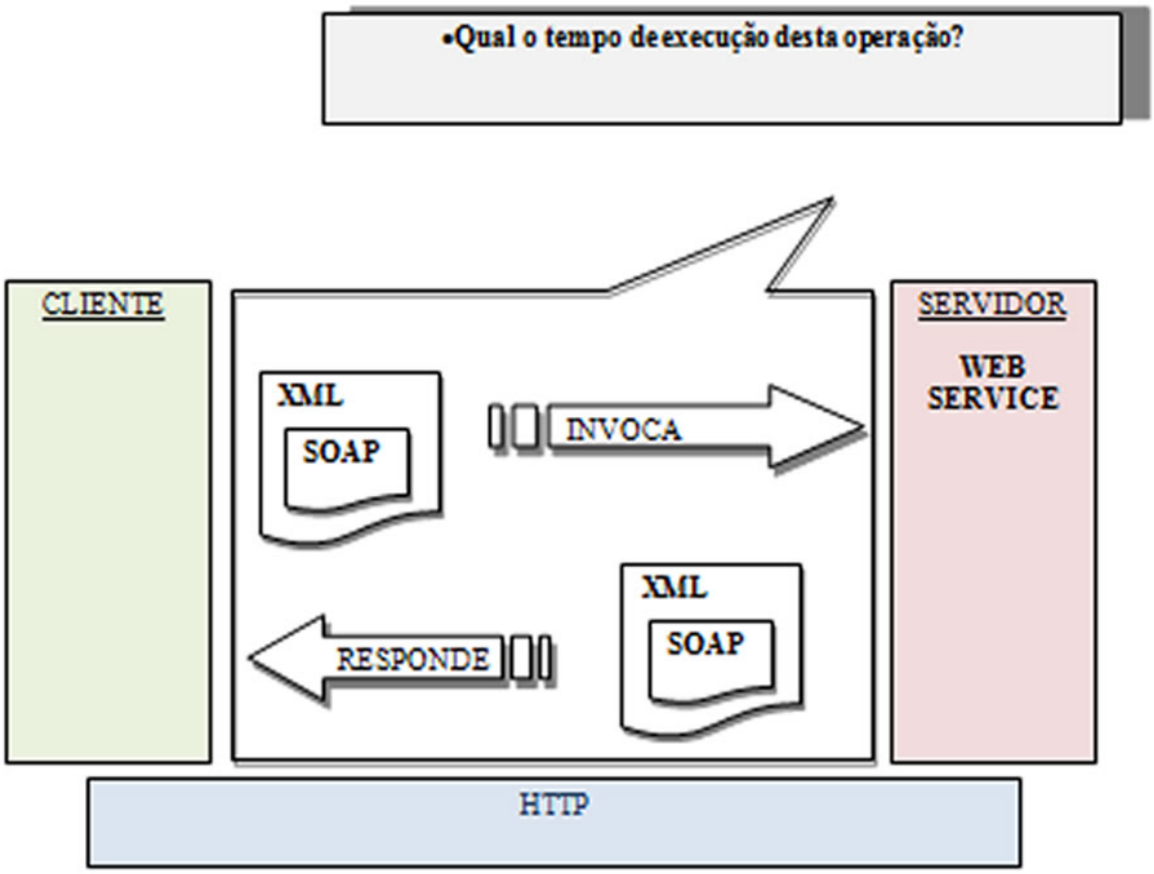

Figura 5 - Arquitectura *web service*

3.2 Protocol Buffer

O *Protocol Buffer* (protobuf) é uma tecnologia desenvolvida pela Google que permite definir de forma simples estruturas de dados, independentemente da linguagem de programação (Google Open Source Blog, 2008). Esta tecnologia pode substituir, em determinadas situações, o tradicional *XML* e segundo a documentação fornecida pela Google, o *protocol buffer* é mais flexível, eficiente e possui mecanismos automáticos de serialização de estruturas de dados, tal como o *XML*, mas mais pequeno, mais rápido e mais simples (Google Code, 2010a).

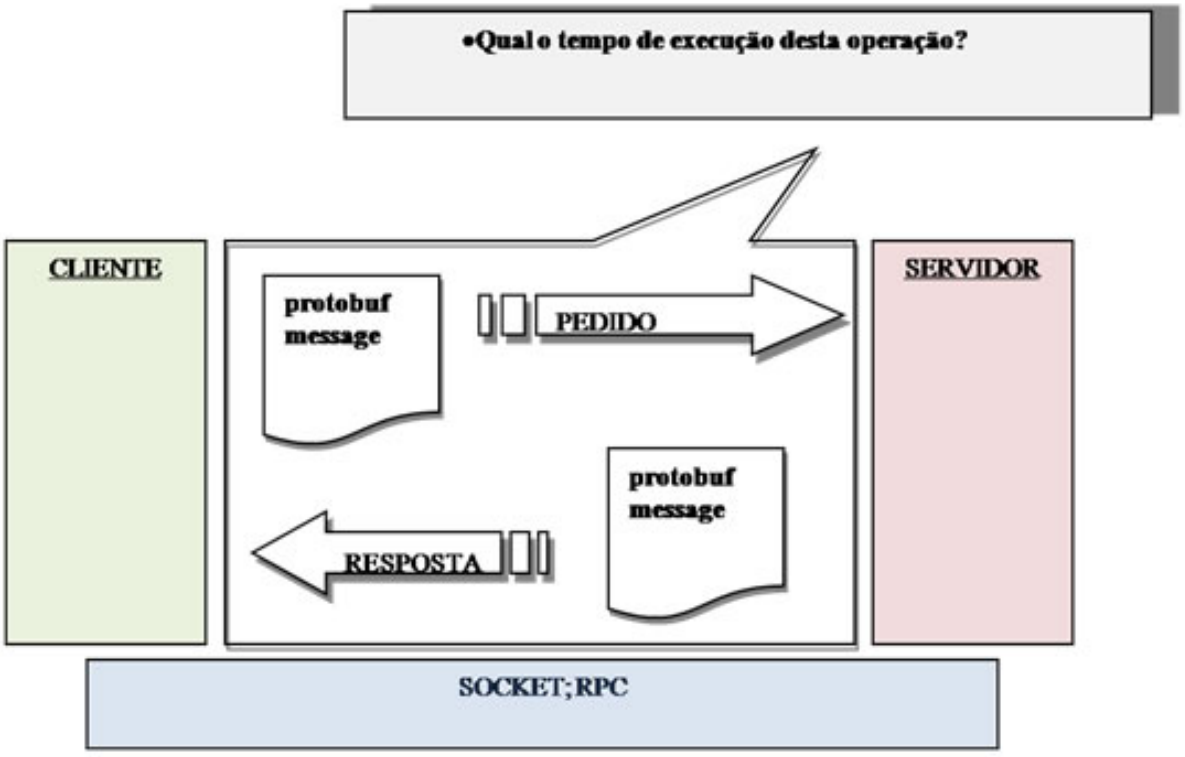

Figura 6 - Implementação do *protobuf*

3.3 Cenários de teste

Neste estudo pretende-se analisar o comportamento das tecnologias *web services* e *protocol buffer* na comunicação de informação entre cliente/servidor. Para o efeito, desenvolveu-se, para cada tecnologia, uma aplicação que simula a realização de uma simples operação de envio de dados pessoais (identificador, nome e email). O cliente envia um conjunto de pedidos para o servidor e este responde com um valor inteiro, sendo a variável tempo o parâmetro em análise. A Figura 7 mostra a sequência de processos afectos o modo de funcionamento da aplicação cliente.

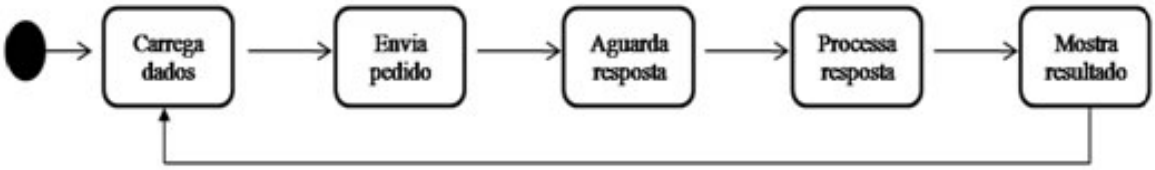

Figura 7 – Sequência de processos

Esta análise é conseguida com o desenvolvimento de dois cenários de teste:

- Cenário 1 – Aplicação cliente executa num dispositivo móvel;
- Cenário 2 – Aplicação cliente executa num computador pessoal.

O cenário 1 analisa o tempo de resposta entre a aplicação que executa num dispositivo móvel e o servidor e o cenário 2 analisa o mesmo parâmetro com a diferença que a aplicação cliente executa num computador pessoal. As Figuras 8 e 9 mostram os vários cenários de teste efectuados.

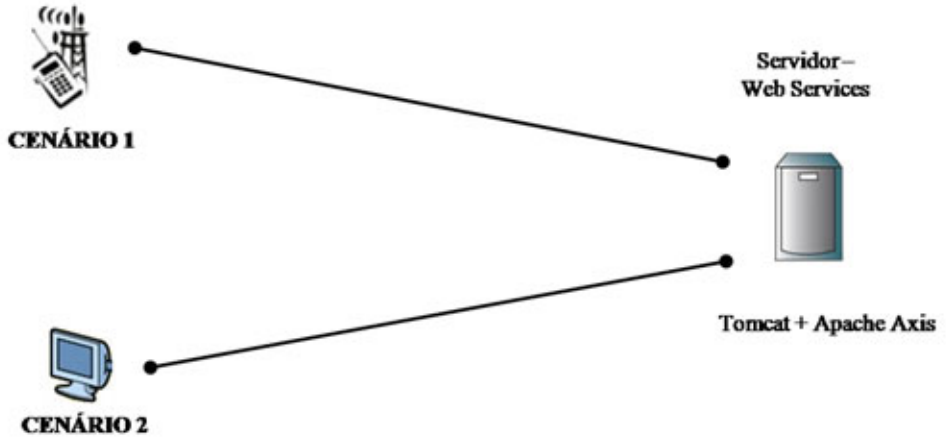

Figura 8 – Cenário de teste *web services*

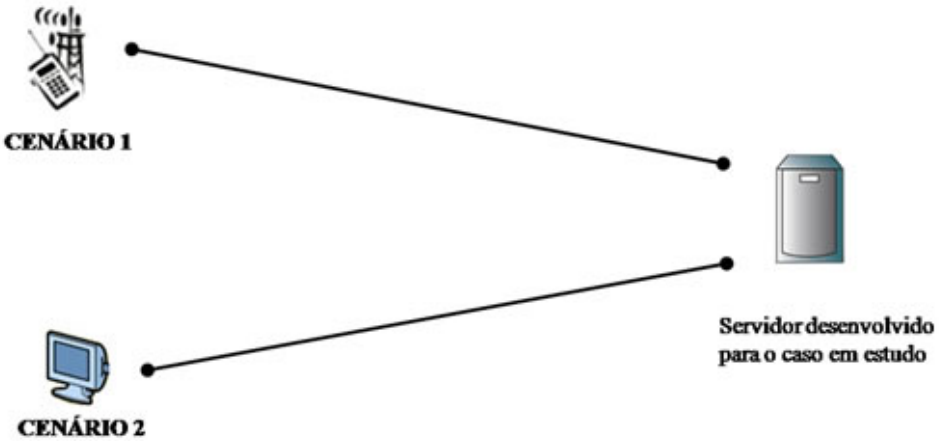

Figura 9 – Cenário de teste *protocol buffer*

Os testes foram realizados recorrendo à rede de dados do ISEP (intranet) e a uma placa 3G (placa banda larga TMN) procurando, desta forma, criar um cenário de teste muito próximo da realidade. Foi disponibilizado, pelo ISEP, um servidor onde se procedeu à virtualização do sistema operativo *Win Server 2003* e onde foram instaladas as aplicações *Tomcat + Apache Axis* para os *web services* e as aplicações desenvolvidas com recurso ao *protocol buffer* (Apache Software Foundation, 2010). Como dispositivo móvel, recorreu-se ao emulador que incorpora a ferramenta de desenvolvimento *NetBean 6.7.1*.

3.3.1 Aplicações de teste de Web Services

O desempenho dos *web services*, em ambos os cenários de teste, foi registado a partir do desenvolvimento, em *Java*, de duas aplicações cliente. Uma desenvolvida para executar em dispositivos móveis (*J2ME*) e outra para executa num ambiente típico de cliente/servidor (*J2SE*). Estas duas aplicações são a seguir descritas.

Serviço

O serviço, desenvolvido em *Java*, recebe o pedido do cliente com os dados pessoais de determinado individuo e devolve um valor inteiro, sendo executado num servidor *Tomcat* 5.5 com a *Framework Apache Axis*.

WSClient J2ME

Esta aplicação foi desenvolvida com recurso à plataforma *J2ME*, a qual apresenta algumas limitações na utilização do protocolo *SOAP* e do *XML*, que são a base de funcionamento da tecnologia *web services*. Para colmatar essa limitação utilizou-se dois projectos, o *KSOAP* e o *KXML* (Knudsen, 2002) (SourceForge, 2006). O código a seguir apresentado, mostra a implementação de um *web service* na tecnologia *J2ME*.

```
…
public void testWebService() throws Exception {
    …
    // endereço do serviço
    String url = "http://localhost:8080/axis/Person.jws";
    SoapObject client;

    // Cria protocolo de transporte HTTP
    HttpTransport ht = new HttpTransport(url,"SearchPerson");

   while(n<100){
        // Cria objecto SOAP
        client = new  SoapObject(url,"SearchPerson");

        // Adiciona parâmetros
        client.addProperty("id",new Integer(10));
        client.addProperty("nam",new String("Ali John Moises"));
        client.addProperty("email",new String("alijohn@mail.com"));

        // Invoca o web service e mostra o resultado final
        System.out.println(ht.call(client));
        n++;
    }
    timeFinal = System.currentTimeMillis();
     System.out.println("Iterações: "+n+" --> Tempo Final: "+(timeFinal-timeStart)/1000);
 }
…
```

WSClient J2SE

A aplicação foi desenvolvida na plataforma *J2SE*. Para utilizar o *web service*, localizado no servidor *Tomcat*, foi necessário adicionar à aplicação as bibliotecas *axis.jar*, *commons-discovery-0.2.jar*, *commons-logging-1.0.4.jar*, *jaxrpc.jar*, *saaj.jar*, *wsdl4j-1.5.1.jar*. O código a seguir apresentado, mostra a implementação de um *web service* na tecnologia *J2SE*.

```
…
public static void main(String[] args) {
        …
         // endereço do serviço
         String urlWS = "http://localhost:8080/axis/Person.jws";

         // Cria os parâmetros de entrada para o web service
         Object[] params = {new Integer(1), new String("Ali John Moises"), new
                              String("alijohn@mail.com")};

         // O objecto Service possibilita a comunicação com o web service
         Service service = new Service();

          // O objecto Call é responsável por invocar o web service
         Call call = (Call) service.createCall();

          // Associa o objecto Call ao url do web service
         call.setTargetEndpointAddress(urlWS);
         timeStart = System.currentTimeMillis();
         while(n<100){
            call.setOperationName("SearchPerson");

           // Invoca o web service
            Integer ret = (Integer) call.invoke(params);
            System.out.println("Resultado: " + ret);  n++;
         }
         timeFinal = System.currentTimeMillis();
         System.out.println("Iterações: "+n+" --> Tempo Final: "+(timeFinal-
                              timeStart)/1000);
       }
```

3.3.2 Aplicações de teste do Protocol Buffer

O projecto inicial, denominado *protocol buffer*, foi idealizado com a finalidade de optimizar as estruturas de dados utilizadas na troca de informação entre plataformas, onde não eram contempladas as aplicações desenvolvidas para dispositivos móveis. No entanto, foi desenvolvida uma versão do *protocol buffer* para *J2ME*, que implementa as principais funcionalidades da versão original, desenvolvida para *J2SE*, existindo compatível entre ambas (Google Code, 2009b).

A implementação para *J2ME* do referido protocolo, denominado *protobuf-javame*, teve em consideração algumas limitações que os dispositivos móveis apresentam, como por exemplo, a autonomia, a capacidade de processamento e a taxa de transmissão. Partindo destas restrições, a sua implementação tem de ser leve, para não consumir muitos recursos, e pequena, para não ocupar muito espaço. Como se trata de um projecto bastante recente é possível que esta implementação venha a sofrer alterações (Google Code, 2009b).

A implementação do *protocol buffer* pressupõem o seguimento de determinadas regras, nomeadamente (Google Code, 2010c):

- Definição da estrutura de dados num ficheiro *.proto;*
  - **J2SE**

```
option java_package = "<designação>";
option java_outer_classname = "<designação>";

 <designação>{
   required int32 value1 = 1;
   required int32 value2 = 2;
   …
}
 <designação>{
          required int32 result = 1;
    …
}
```

  - **J2ME**

```
option java_package = "<designação>";

 <designação>{
          optional int32 value1 = 1;
          optional int32 value2 = 2;
         …
   }
```

- Compilar o código *Java* a partir do ficheiro *.proto*, utilizando respectivamente:
  - *Protocol Buffer* → compilador *proto*

    protoc -I=$SRC_DIR --java_out=$DST_DIR $SRC_DIR/<nome_fich>.proto;
  - *Protocol Buffer J2ME* → ficheiro *proto2javame.jar*

    java –jar proto2javame --java_out=$DST_DIR $SRC_DIR \<nome_fich>.proto.
- Adicionar o *output* gerado a partir do ficheiro <nome_fich>*.proto* ao projecto;
- Adicionar o *package com.google.protobuf* e o ficheiro *protobuf-javame.jar* aos projectos desenvolvidos em *J2SE* e *J2ME*, respectivamente.

PBClient e PBServer

A aplicação cliente (PBClient) implementa a funcionalidade de envio de dados pessoais de um determinado indivíduo e a aplicação servidor (PBServer) responde com um valor inteiro. A implementação do *protobuf-javame* não permite definir mais do que uma mensagem no ficheiro *.proto* (Google Code, 2009b) e caso sejam definidas mais mensagens, estas são simplesmente ignoradas pelo compilador. A mensagem definida para *J2ME* apresenta a seguinte estrutura:

```
java_package = "tese.search.person";

Person {
        optional string name = 1;
        optional int32 id = 2;
        optional string email = 3;
        optional int32 result = 4;
}
```

Ao contrário do *protobuf-javame*, a implementação padrão permite definir, na mesma estrutura, mais que uma mensagem. A estrutura da mensagem definida para a implementação desenvolvida em *J2SE*, apresenta a seguinte estrutura:

```
package tutorial;
option java_package = "com.search.person";
option java_outer_classname = "SearchPerson";

 Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

}


 Request{
        required int32 result = 1;
}
```

Para este exemplo em concreto, a estrutura de código para a aplicação cliente e para a aplicação servidor são iguais, independentemente da tecnologia utilizada (J2ME e J2SE). As estruturas de código são de seguir representadas.

PBClient

```
Person.Builder p;
Person pReq, pRes;

while(n<100)
   {
      // cria mensagem
      p = Person.newBuilder();
      p.setId(1);
      p.setName("Ali John Moises");
      p.setEmail("alijohn@mail.com");
      pReq = p.build();

      // envio da mensagem
      pReq.writeDelimitedTo(outputStream);
      outputStream.flush();

      // recepção da mensagem
      pRes = Person.parseDelimitedFrom(inputStream);
      System.out.println(pRes.getResult());
      n++;
   }
```

PBServer

```
Person person;
while(true)
   {
      // lê mensagem
      person = Person.parseDelimitedFrom(inputStream);

      // mostra conteúdo da mensagem
      System.out.println("***Person****");
      System.out.println("ID: "+person.getId());
      System.out.println("Name: "+person.getName());
      System.out.println("Email: "+person.getEmail());
      System.out.println("********************");

      // espera
      for (int i=0; i<50; i++){
      }

      // cria mensagem resposta
      Person.Builder req = Person.newBuilder();
      req.setResult(2);

      // envia mensagem
      req.build().writeDelimitedTo(outputStream);
      outputStream.flush();
   }
```

3.3.3 Resultados / Conclusão

As medições recolhidas têm por base o tempo que cada uma das tecnologias demorou a realizar o ciclo de iterações proposto. Não foi contabilizado o tempo de estabelecimento da conexão com o servidor remoto. A Tabela 1 mostra os resultados obtidos.

Tabela 1 – Resultados protocol buffer vs web services

Tecnologia	**Iterações**	***WEB SERVICES***		***PROTOCOL BUFFER***	
		Tempo de Execução (sec)		**Tempo de Execução (sec)**	
		Intranet	**Internet – Placa TMN G3**	**Intranet**	**Internet – Placa TMN G3**
J2ME	10000	Erro	----	≅ 35sec	----
	1000	≅3min49sec	----	≅4sec	----
	500	≅21sec	----	≅2sec	----
	100	----	≅27 sec	---	≅10 sec
J2SE	10000	≅1min24sec	----	≅29sec	----
	1000	≅9sec	----	≅3sec	----
	500	≅5sec	----	≅1sec	----
	100	----	≅28 sec	----	≅9 sec

Os resultados observados permitem verificar que o *protocol buffer* é mais rápido na transmissão de dados que os *web services* e essa diferença é mais significativa na tecnologia *J2ME*. Estes resultados sustentam a informação que a Google disponibiliza na página dedicada ao projecto *protocol buffer*, relativamente à maior velocidade de transmissão de dados do *protocol buffer* em relação aos *web services* (Google Code, 2010a). Esta diferença pode ser explicada pelo facto das mensagens geradas no *protocol buffer* apresentarem um tamanho mais reduzir que as mensagens produzidas na tecnologia *web services*. Os *web services* utilizam para a transmissão de mensagens, a tecnologia *XML*, que incorpora um conjunto da *tags* que aliadas ao conteúdo da mensagem, aumenta consideravelmente a quantidade de informação a transmitir.

Os resultados mostram ainda que o *protocol buffer* associado à tecnologia G3 funciona e com um desempenho bastante satisfatório quando comparado como os *web services*.

Ao contrário dos *web services*, o *protocol buffer* é uma tecnologia bastante recente, existindo por isso uma grande margem de evolução. Face aos resultados demonstrados, adivinha-se que esta tecnologia venha a ter grande sucesso na área da transmissão de dados.

A implementação do *protocol buffer*, para *J2ME*, apresenta ainda algumas limitações em relação à implementação para *J2SE*. As perspectivas são, no entanto, boas tendo em conta os resultados apresentados. Neste momento, o principal objectivo desta implementação é procurar resolver as limitações que ainda possam existir, como por exemplo, a possibilidade de enviar mais que uma mensagem em simultâneo.

Os *web services* apesar das suas limitações são actualmente, a tecnologia com maior aceitação no mercado onde existe partilha de serviços. As provas de fiabilidade dadas asseguram-lhe esse reconhecimento.

Um aspecto importante para o sucesso dos *web services* é o facto de recorrer à tecnologia *XML* para troca de mensagens. O *XML* é uma linguagem padrão que confere interoperabilidade entre diferentes plataformas. Uma outra vantagem dos *web services* é o facto destes, na esmagadora maioria dos casos, utilizarem como protocolo de transporte o *HTTP* e desta forma as restrições de acesso impostas por *firewalls* deixam de existir.

No projecto em estudo, optou-se por implementar o *protocol buffer* em virtude de este apresentar melhor desempenho quando comparado com os *web services*. Um outro aspecto relevante para a utilização da referida tecnologia, está relacionado com a forma de estruturação da informação a enviar, as quais são caracterizadas por serem flexíveis e simples de implementar.

4. Segurança

A informação é o veículo utilizado pelo Homem para descrever situações envolventes, como objectos, cenários, pessoas, etc. A importância da informação é tão grande que levou as sociedades a criarem mecanismos para a sua gestão e distribuição.

As tecnologias de informação vieram revolucionar a forma como o acesso à informação é realizado. Através das novas tecnologias, o acesso tornou-se um processo permanente e rápido. No entanto, aliado a estas mais-valias, tornou-se mais premente o problema relacionado com a segurança. A informação passa a estar num estado de vulnerabilidade, permitindo o acesso por parte de indivíduos não autorizado. As principais vulnerabilidades residem no acesso aos sistemas onde a informação está armazenada e na comunicação destes para a aplicação cliente.

Usualmente, no acesso a repositórios de informação, é utilizado uma conta de utilizador que é constituída usualmente por um *login* e uma *password.* Os dados referentes à conta são previamente registados no sistema e a comparação dos mesmos com os dados introduzidos pelo utilizador, valida ou invalida, o acesso à informação. Este processo de validação é conhecido como autenticação, apresentando algumas fragilidades relacionadas com a segurança. Os dados utilizados neste processo podem ser facilmente descobertos, recorrendo a técnicas como:

- *Phishing* – Este processo consiste em obter dados pessoais, usando a identidade de entidades credíveis;
- *Sniffing* – Quando a informação circula em redes não seguras, o recurso a aplicativos capazes de capturar o tráfego permite ter acesso a informação confidencial;
- Visualização da digitação dos dados.

Outra forma indevida de ter acesso a informação confidencial está no processo de transmissão de um agente para outro. Durante o processo de transporte, a informação pode ser capturada, usando a técnica de *sniffing*, permitindo ao intruso visualizar e manipular os dados. Este tipo de ataque ocorre sobretudo quando a comunicação opera em canais não seguros. Torna-se, por isso, necessário implementar mecanismos de segurança que minimizem o estado de vulnerabilidade da informação, garantindo que os conceitos de integridade, autenticidade e confidencialidade são preenchidos.

4.1 Comunicações Seguras

No sentido de satisfazer os conceitos subjacentes à segurança da informação, referidos anteriormente, foram desenvolvidos ambientes de comunicação seguros, oferecendo ao seu utilizador garantias que a informação, por si transmitida, circula num canal de comunicação seguro.

Foram projectados e implementados alguns protocolos de segurança, como o *Transport Layer Security/Secure Sockets Layer* (*TLS/SSL*), que têm como principal objectivo conferir segurança na comunicação da informação, conforme mostra a Figura 10. Os protocolos operam na camada 7 do modelo OSI[3] e recorrem à criptografia para atingir o seu objectivo (Wikipédia, 2010d).

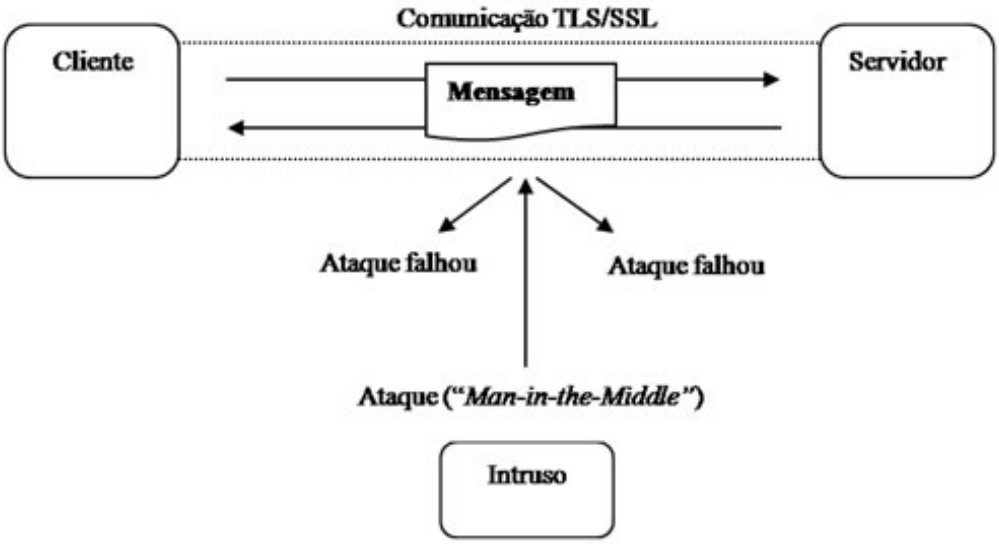

Figura 10 – Comunicação Segura *TLS/SSL*

Estes protocolos utilização algoritmos criptográficos que envolvem a execução de operações aritméticas complexas, as quais requerem um esforço extra de processamento, e para as quais a maioria dos dispositivos móveis não está actualmente preparada.

4.2 Criptografia

A criptografia deriva da criptologia, ramo da Matemática e o termo resulta das palavras gregas "kyptós" e "gráphein", que significam "escondido" e "escrita" (Wikipédia, 2010a). Esta ciência envolve um conjunto de conceitos e técnicas que têm como objectivo transformar a informação original em informação ilegível, de forma a garantir que só o emissor e o receptor possam ter acesso à mesma.

[3] *Open System Interconncetion*

A história diz-nos que a criptografia sempre foi um campo muito explorado, normalmente associado a actividades militares e diplomáticas. Na época dos Romanos, Júlia César cifrava os seus despachos, trocando as letras originais por outras letras, como por exemplo, A por D, escondendo dos seus inimigos a informação original. Actualmente, o objectivo principal continua a ser o mesmo, ou seja, assegurar que só os indivíduos devidamente autorizados têm acesso à informação (Anderson, 2008).

4.2.1 Cifras

As cifras são utilizadas para manter os dados secretos, transformando dados legíveis (*plaintext*) em dados ilegíveis (*ciphertext*) e vice-versa, conforme mostra a Figura 11.

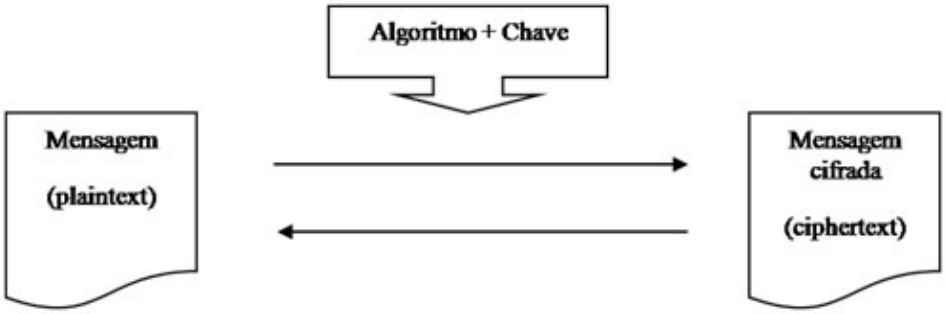

Figura 11 - Processo de codificação e descodificação de mensagens

As cifras criptográficas utilizam chaves para gerar *ciphertext* únicos. A transformação de dados cifrados em texto legível é possível com a mesma chave ou com uma chave correspondente, chaves simétricas e chaves assimétricas, respectivamente (Sun Microsystems, 2004b). Juntamente com as chaves são utilizados algoritmos matemáticos para cifrar e decifrar dados, sendo o processo constituído por várias partes, nomeadamente a mensagem original, o algoritmo criptográfico, a chave criptográfica e a mensagem cifrada (Anderson, 2008).

4.2.2 Algoritmo de Criptografia

Algoritmo pode ser definido, de forma generalista, com sendo um conjunto de regras que visam solucionar um problema. Neste caso especifico, um algoritmo é um conjunto de fórmulas matemáticas que aplicadas repetidamente sobre uma mensagem, elevam-na a um estado de ilegibilidade. Os algoritmos criptográficos são caracterizados por serem públicos e por terem associados chaves criptográficas. A robustez destes algoritmos é tanto maior quanto maior for o tamanho da chave criptográfica utilizada e a complexidade do próprio algoritmo (Anderson, 2008).

4.2.3 Chaves Criptográficas

As chaves criptográficas, que são o segredo das mensagens cifradas, apresentam um conjunto de variáveis usadas como parâmetros nos algoritmos criptográficos, controlando o processo de transformação de texto legível em texto ilegível. O tamanho da chave é configurável, conforme as necessidades de segurança (Anderson, 2008).

4.2.4 Cifra Simétrica

No sistema de cifra de chave simétrica, existe uma chave secreta partilhada pelos intervenientes que é usada para cifrar e decifrar as mensagens trocadas. A existência de uma chave secreta partilhada, pode ser entendido como uma falha de segurança, pois qualquer indivíduo que descubra o segredo tem acesso à mensagem cifrada. Outro problema reside na distribuição da chave, uma vez que os canais utilizados na distribuição são inseguros, sendo necessário recorrer a técnicas que aumentam a segurança, como por exemplo, o *TLS*. A utilização de uma chave secreta, por cada par de utilizadores, pode dificultar o mecanismo de gestão de chaves.

Apesar das limitações apresentadas, este sistema é bastante utilizado para cifrar e decifrar dados. Adoptando-se procedimentos que minimizem as limitações mencionadas, o recurso a chaves simétricas apresenta como principal vantagem o facto de ser um sistema com um bom desempenho em termos de rapidez e difícil de quebrar quando bem implementado. Com a implementação deste sistema, atinge-se um dos pilares subjacentes à segurança, a confidencialidade.

A cifra simétrica é implementada por alguns algoritmos relevantes, dos quais destacou-se o *Data Encryption Standard* (*DES*) e o *Advanced Encryption Standard* (*AES*).

DES

O algoritmo foi criado na década de 70 pela IBM. Nessa mesma década, os Estados Unidas da América sentiram a necessidade de adoptar um padrão para a codificação de dados considerados sensíveis. O *National Bureau of Standards* (NBS), agora *National Institute of Standards and Technology* (NIST), ficou responsável por tornar o *DES* padrão na codificação de dados. O algoritmo cifra simetricamente a informação em blocos de 64 bits com recurso a uma chave de 56 bits. Actualmente, este algoritmo é considerado inseguro, sendo a sua utilização desaconselhada (RSA Laboratories, 2010).

AES

As vulnerabilidades apresentadas pelo *DES* levaram a NIST a anunciar, em 2002, o *AES* como novo padrão de codificação de dados. O algoritmo, também conhecido como *Rijndael*, é um

cripto-sistema simétrico baseado na cifra de blocos que utiliza uma chave de tamanho variável, 128, 192 ou 256 bits, respectivamente, ao contrário do *DES* que utiliza uma chave de 56 bits. Actualmente, o *AES* é considerado um algoritmo seguro sendo utilizado pelo EUA como padrão de codificação de dados (RSA Laboratories, 2010).

4.2.5 Cifra Assimétrica

O sistema de cifra assimétrica, também conhecido como chave pública, caracteriza-se por envolver um par de chaves, uma pública e outra privada. A chave pública é do conhecimento de todos, enquanto a chave privada permanece secreta. Para melhor descrever o processo de funcionamento suponhamos o cenário onde **A** e **B** trocam mensagens:

- **A** cifra a mensagem com a sua chave privada e envia-a a **B**;
- **B** utiliza a chave pública de **A** para decifrar a mensagem;
- **B** cifra a mensagem resposta com a sua chave pública e envia-a a **A**;
- **A** decifra a mensagem resposta com sua chave privada.

Ao contrário do sistema de cifra simétrica, este garante confidencialidade e autenticidade, quando combinando a chave privada do emissor com a chave pública do receptor, no envio de uma mensagem. No processo de criação do par de chaves assimétricas, são envolvidos procedimentos matemáticos complexos que exigem maior capacidade de processamento, tornando o sistema mais lento que o simétrico. Este sistema, aparentemente seguro, apresenta algumas vulnerabilidades, nomeadamente quando se utiliza a técnica *man-in-the-middle*. Nesta técnica um terceiro agente introduzir-se no meio de uma comunicação, fazendo-se passar pelo destinatário das mensagens.

O sistema de cifra assimétrica é frequentemente utilizado para ultrapassar uma limitação do sistema de cifra simétrica, designadamente a distribuição da chave secreta. Este método cria um ambiente seguro, capaz de garantir segurança na partilha do segredo.

Alguns exemplos de algoritmos que implementam este tipo de sistema são, o Rivest, Shamir e Adleman (*RSA*), o *Digital Signature Algorithm* (*DSA*) e o El Gamal, Diffie-Helman. De entre o conjunto de algoritmos criptográficos apresentados, analisou-se com particular a atenção o *RSA*.

RSA

O algoritmo *RSA* foi desenvolvido por Ronald Rivest, Adi Shamir, and Leonard Adleman em 1977 e introduz o conceito de chave pública. O par de chaves, pública e privada, é gerado com recurso à teoria elementar dos números, em particular os números primos. O *RSA* apresenta-se como um algoritmo seguro e difícil de ser quebrado. O nível de segurança depende do tamanho da chave, sendo

actualmente recomendado pelo *RSA Laboratories*, o uso de um tamanho de 1024 bits para informação sensível e 2048 bits para informação extremamente sensível (RSA Laboratories, 2010).

4.2.6 Cifra Híbrida

Um sistema baseado em cifra híbrida consiste na combinação de cifra simétrica e cifra assimétrica. A combinação das duas técnicas permite implementar um sistema híbrido, aproveitando desta forma as vantagens de ambas as abordagens. O sistema de cifra simétrica é normalmente empregue para transmitir dados cifrados, enquanto o sistema baseado em cifra assimétrica é utilizado para garantir segurança na distribuição da chave secreta (cifra simétrica). Um protocolo muito conhecido que implementa este sistema é o *TLS* (Microsoft Support, 2007). Para melhor descrever o processo de funcionamento, suponhamos o cenário onde **A** pretende criar um canal seguro para enviar mensagens a **B**:

- **A** envia a chave pública a **B**;
- **B** gera a chave secreta (chave de sessão), cifra-a com chave pública de **A** e envia a **A**;
- **A** decifra a chave de sessão com a sua chave privada;
- **A** e **B** trocam mensagens cifradas com a chave de sessão.

4.2.7 Modo de Operação

Um factor importante na implementação de um algoritmo criptográfico, reside no modo de operação, que especifica a forma como a mensagem original é cifrada e posteriormente decifrada. Neste sentido, analisou-se dois modos de operação: por blocos (*block cipher*) e em cadeia (*stream cipher*).

O modo de operação por blocos consiste em dividir, em blocos de *bytes* de tamanho fixo, a mensagem original. Este modo suporta ainda dois modos diferentes de operação, o *electronic codebook* (*ECB*) e o *cipher-block chaining* (*CBC*). O *ECB* consiste em dividir a mensagem original em blocos que posteriormente são cifrados separadamente. O *CBC*, mais complexo que o anterior, combina o bloco da mensagem original a codificar nesse instante, com o resultado da mensagem cifrada gerado na etapa anterior. Este modo de operação requer um vector de inicialização que combina com o primeiro bloco da mensagem original a cifrar (Anderson, 2008).

O modo de operação em cadeia caracteriza-se pela transformação individual de cada *bit* ou *byte*. Esta abordagem, à semelhança do modo de operação por blocos, também se divide em dois modos diferentes de operação, o *cipher feedback* (*CFB*) e o *output feedback* (*OFB*). Estes dois modos têm um modo de operar semelhante ao *CBC*, ou seja, a cifra de um bloco depende dos blocos

anteriores (função de realimentação). A principal diferença entre o modo *OFB* e *CFB* reside no facto do primeiro utilizar o *output* gerado (chave continua) anteriormente para aplicar na próxima operação de codificação, enquanto o segundo utiliza a mensagem cifrada gerada no processo anterior (Anderson, 2008).

4.2.8 Funções Hash

As funções de *hash* permitem construir, a partir de uma mensagem a transmitir (*input*) de tamanho variável, um *output* de tamanho fixo, usualmente mais pequeno, conhecido como valor de *hash* ou *digest* (Figura 12). A partir do *output* gerado é matematicamente inviável determinar o *input* inicial, sendo estas funções também designadas de *one-way function* (Anderson, 2008).

Figura 12 - Funcionalmente da função de *hash*

As mensagens *digest* são particularmente úteis para assegurar que os dados não são alterados quando transferidos de um ponto para outro (Wikipédia, 2010b). O potencial da utilização desta técnica resulta da sua combinação com outras técnicas. Uma implementação prática das funções de *hash* é o protocolo de segurança *TLS* versão 1.2, que utiliza na implementação de canais de comunicação seguros os seguintes algoritmos de *hash*: *MD5, SHA-1, SHA-1,SHA-224, SHA-256, SHA-384 e SHA-512* (RFC5246, 2008).

4.3 Segurança em Sistemas de Saúde Móveis

As tecnologias de informação e comunicação proporcionaram aos profissionais de saúde maior rapidez no acesso a dados clínicos, garantindo desta forma maior eficácia nos serviços de saúde prestados. Surgiram aplicações específicas para a área da saúde, onde os dados clínicos são centralizados, podendo ser acedido a partir de um computador pessoal. Com o avanço das tecnologias móveis, a informação passou a estar sempre (ou quase sempre) disponível, independentemente da localização geográfica do receptor.

Num sistema de informação clínico, a segurança é um ponto muito importante que deve ser alvo de um estudo pormenorizado. Os dados partilhados são extremamente sensíveis, existindo a necessidade de implementar mecanismos de segurança robustos. Se num sistema tradicional de

cliente/servidor, a solução usual passa pela implementação de canais de comunicação seguros (*TLS*), nos dispositivos móveis este compromisso é mais difícil de atingir, uma vez que estes apresentam algumas limitações, nomeadamente autonomia, capacidade de processamento e taxa de transmissão de dados, dificultando a implementação de mecanismos de segurança.

4.3.1 Protocolo de Segurança

A necessidade de conferir segurança na informação partilhada entre os vários sistemas em estudo, levou a que se optasse por desenvolver um protocolo de segurança que se ajuste aos requisitos fundamentais de segurança pretendidos para o projecto:

- Criação de um canal de comunicação seguro para a troca de informação, nomeadamente os registos médicos e os dados necessários ao processo de autenticação no sistema;
- Garantir que a informação trocada seja unicamente interpretável pelo destinatário pretendido.

Apesar da questão da segurança ser um aspecto fundamental, esta deve ser conciliada com a funcionalidade e desempenho do sistema.

Para atingir os requisitos pretendidos devem ser cumpridos os seguintes critérios:

- O servidor deve conter um registo de todos os dispositivos autorizados a aceder aos serviços. Cada registo deve relacionar um IDCliente, um IDServidor, um IDSessão e um código (*IMEI* ou *Pin Code*). Estes ID´s são actualizados periodicamente pelo servidor e enviados ao cliente;
- O cliente deve conter um registo que associa o seu IDCliente com o IDServidor e o IDSessão;
- A utilização de sessões é parte integrante do protocolo de segurança proposto;
- As mensagens enviadas com os pedidos ou respostas devem ser estruturadas de forma a fornecer informação ambígua em caso de ataque. A utilização de identificadores (ID´s) para identificar pacientes produz o efeito de ambiguidade desejado. A estruturação de mensagens é descrita no Capítulo 5.

À semelhança do protocolo *TLS* (Microsoft Support, 2007) (RFC5246, 2008), também nesta implementação recorreu-se a um sistema de segurança híbrido, ou seja, combinação de cifra assimétrica com cifra simétrica. A cifra assimétrica é utilizada no processo de autenticação, enquanto a

cifra simétrica é utilizada no processo de troca de informação. Com um sistema híbrido, os aspectos fundamentais de segurança ficam assegurados: a cifra assimétrica assegura a criação de um ambiente seguro, ideal para o processo de autenticação e a cifra simétrica é utilizada para garantir confidencialidade da informação.

4.3.2 Sessões

As sessões, no contexto apresentado, são uma ferramenta importante permitindo tornar a aplicação mais funcional. O processo de autenticação requer um consumo elevado de recursos que é evitado com a incorporação de sessões. Cada sessão é composta por um IDSessão, *timeout* e chave secreta. O IDSessão identifica uma sessão activa e o *timeout* é uma variável que possibilita a actualização periódica da chave secreta.

4.3.3 Estrutura do Protocolo de Segurança

O protocolo de segurança está estruturado em duas camadas, nomeadamente a camada **Autenticação** e a camada **Comunicação**. A primeira camada é responsável por estabelecer a comunicação entre os agentes intervenientes, enquanto a segunda camada é responsável pelo tratamento dos dados transferidos.

Camada Autenticação

O processo de autenticação envolve um conjunto de estados que são a seguir descritos:

1. A aplicação cliente envia para o servidor o IDSessão e a lista com os algoritmos criptográficos disponíveis. Caso o cliente tenha uma sessão activa, todo o processo de autenticação foi previamente estabelecido, podendo ocorrer transferência segura de dados (o processo de codificação é realizada com a chave secreta gerada na última sessão). Caso não exista sessão inicia-se o processo de autenticação com o IDSessão a zero (0). A Figura 13 descreve o processo inicial de criação de um canal seguro;

Figura 13 – Estabelecimento de uma sessão

2. O servidor responde com a sua lista de algoritmos criptográficos e o IDSessão;
3. O servidor gera um par de chaves assimétricas. A chave privada fica na sua posse e a chave pública é enviada para o cliente;
4. Paralelamente ao descrito no ponto 3, é gerada a chave secreta, que é cifrada com a chave privada e enviada para o cliente;
5. O cliente decifra a chave secreta com a chave pública. A partir deste momento as mensagens trocadas são cifradas e decifradas com a chave secreta, por questões de rapidez e gestão de recursos;
6. O cliente cifra o seu IDCliente e envia para o servidor;
7. O servidor decifra o IDCliente e valida-o. Caso seja válido cifrar o seu IDServidor e envia para o cliente;
8. O cliente decifra o IDServidor e valida-o. Caso seja válido o cliente cifra o *IMEI* ou *PinCode* e envia para o servidor;
9. O servidor valida as credenciais (*IMEI* ou *PinCode*) e caso sejam válidas, o servidor cria um IDSessão, associa-o ao IDCliente e envia-o ao cliente;
10. O cliente actualiza o IDSessão e responde com uma mensagem de FIM;
11. O servidor responde com mensagem FIM;
12. O processo de autenticação está concluído.

A Figura 14 mostra a informação trocada, entre o cliente e o servidor, ao longo dos vários estados que constituem o processo de autenticação.

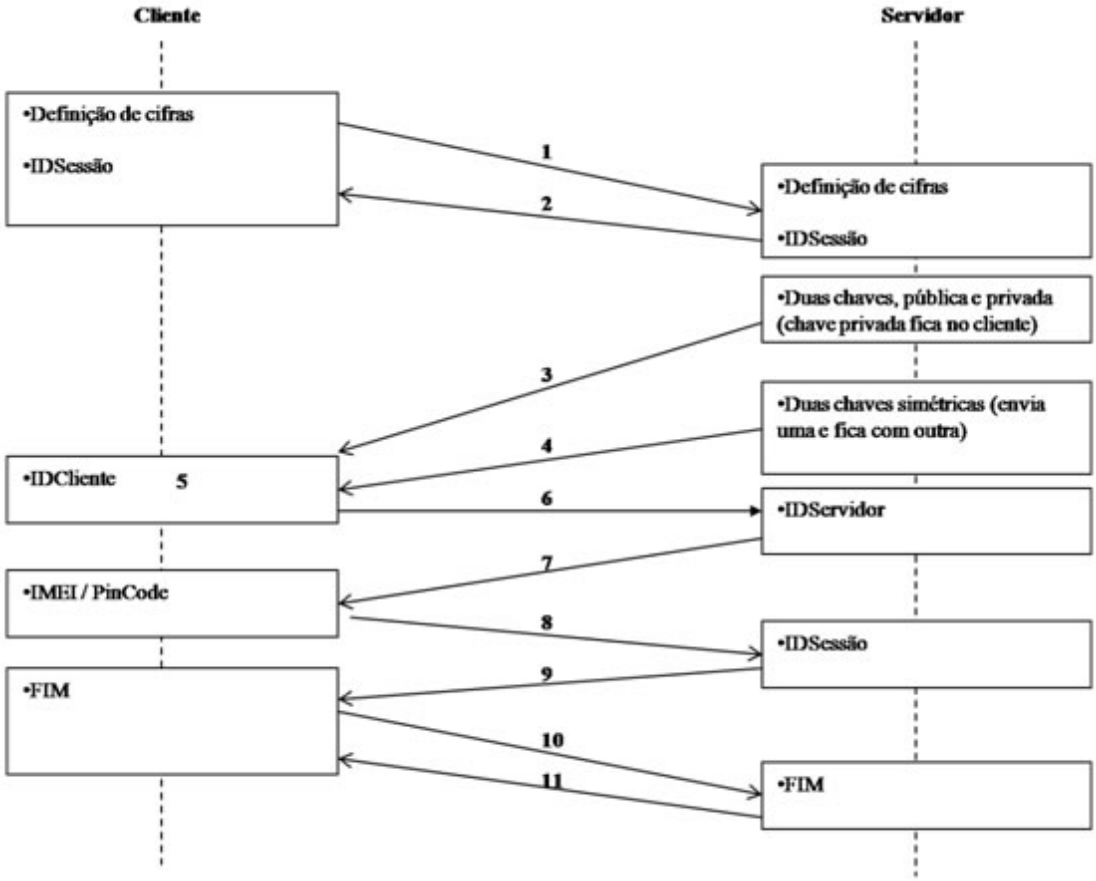

Figura 14 – Informação trocada durante o processo de autenticação

Durante o processo de autenticação é possível que ocorram erros, como por exemplo, erros de validação. Nestas situações o sistema *middleware* envia ao cliente uma mensagem com a especificação do erro. Tendo em atenção as limitações de desempenho das redes móveis e dos dispositivos móveis, o mecanismo de segurança proposto garante um nível de segurança aceitável na transmissão de dados.

Camada Comunicação

A camada de comunicação é responsável por cifrar e decifrar as mensagens transferidas. Após a aplicação do respectivo processo, consoante o caso, o resultado final é encaminhado para a camada responsável pelo tratamento da mensagem.

4.4 Cenários de teste

Após a descrição do mecanismo de autenticação, foram efectuados os respectivos testes. Tal como no Capítulo 3, optou-se por realizar os testes em dois ambientes distintos, o primeiro utilizou a rede de dados do ISEP e o segundo uma placa 3G. O servidor, uma máquina virtual a executar o Win Server 2003, encontra-se nas instalações do ISEP.

4.4.1 Algoritmos de Criptografia

Antes de mencionar os algoritmos de criptografia utilizados, convém alertar para o facto de que não existem cripto-sistemas totalmente seguros, existe sim, um conjunto de variáveis (algoritmo e chave) configuráveis que aumentam o nível de segurança. Uma falha de segurança evidenciada pela grande maioria destes sistemas reside no processo de distribuição das chaves.

O algoritmo utilizado no mecanismo de distribuição da chave secreta foi o *RSA*. A escolha prende-se com o facto de este ser considerado seguro e com uma larga implementação a nível mundial, desde instituições governamentais, universidades, passando por empresas de renome como a Microsoft, Appel, Sun, Novell (RSA Laboratories, 2010).

O *RSA* é tanto mais seguro quanto maior o tamanho da sua chave. No entanto, essa segurança tem um custo em termos de processamento, com o consequente aumento da utilização de recursos (processador e bateria, no dispositivo móvel). Neste projecto, com forte ênfase em dispositivos móveis, o elevado consume de recursos afigura-se como um grave problema. Face a esta realidade, optou-se por implementar todo o processo de criação de chaves (chave secreta e chave pública) no servidor. Foram implementados dois cenários de teste diferentes para o *RSA*, onde foram medidos os tempos de execução para dois tamanhos de chaves, 512 bits e 1024 bits.

Após o processo de distribuição da chave secreta estar concluído, as mensagens a transmitir são cifradas e decifradas, por questões de eficiência, com um algoritmo de cifra simétrica. Neste projecto, utilizou-se o algoritmo *DES* com uma chave de 56 bits e o *AES* com uma chave de 128 bits. Apesar do *DES* ser desaconselhado, foi implementado para permitir a comparação com o *AES*. O *AES* é considerado o sucessor do *DES*, sendo um algoritmo padronizado e seguro.

4.4.2 Tecnologia de suporte

O mecanismo de autenticação foi implementado em *J2ME* utilizando o *package Security and Trust Services APIs (SATSA) for Java™ 1.0*. Este *package*, presente no *MIDP 2.0*, permite estender os recursos de segurança para a plataforma *J2ME* através da adição de quatro *API´s* (Sun Microsystems, 2004b):

- ***SATSA-APDU*** e ***SATSA-JCRMI*** – permitem que as aplicações comuniquem com *smart card*;
- ***SATSA-PKI*** – permite que as aplicações utilizem o *smart card* para assinaturas digitais e gestão de certificados de utilizador;
- ***SATSA-CRYPTO*** **–** confere às aplicações a capacidade de gerar *digest*, verificar assinaturas digitais e cifrar e decifrar dados.

As mensagens inerentes a todo o mecanismo de autenticação foram cifradas e decifradas com recurso à *API SATSA-CRYPTO***.** A troca de mensagens cifradas foi realizada com recursos à estrutura de dados definida na tecnologia *protocol buffer* e estão documentadas no Capítulo 5.

4.4.3 Aplicações de suporte

Para simular o processo de autenticação foram implementadas duas aplicações, a aplicação cliente e a aplicação servidor. Estas duas aplicações são suportadas por um conjunto de *classes*, apresentadas na Tabela 2, que permitem efectuar as várias etapas do processo de autenticação.

Tabela 2 – Classes de suporte ao processo de autenticação

***Classes* de Suporte**		**Descrição**
J2SE	***J2ME***	
RSA	RSAJ2ME	Estas *classes* são responsáveis por cifrar e decifrar mensagens segundo o respectivo algoritmo.
AES	AESJ2ME	
DES	DESJ2ME	
MsgAutenticacao	MsgAutenticacao	Esta *classe* permite gerar as mensagens a enviar ao destinatário.

De salientar que o resultado gerado, após a aplicação do respectivo algoritmo, encontra-se em *bytes*. Os *bytes* são convertidos em hexadecimal e enviados no formato de *String* através do *protocol buffer*.

A seguir, são apresentadas duas estruturas de código que mostram o funcionamento da aplicação servidor e da aplicação cliente para a combinação dos algoritmos *RSA* e *AES*. Relativamente à combinação *RSA* e *DES*, o modo de funcionamento é semelhante.

Servidor - AES

```
. . .
DataOutputStream outputStream = new DataOutputStream(clientSocket.getOutputStream());
DataInputStream inputStream = new DataInputStream(clientSocket.getInputStream());

Autenticacao.C1 msgC1;
Autenticacao.C2 msgC2;
Autenticacao.C3 msgC3;
Autenticacao.C4 msgC4;

Mensagens msg = new Mensagens();
RSA rsa = new RSA(512);
AES aes = new AES(128);

// recebe msgC1
msgC1 = Autenticacao.C1.parseDelimitedFrom(inputStream);
System.out.println("Algoritmos = "+msgC1.getAlg(1));

// envia msgS1
msg.enviaMsgS1(outputStream);

//envia msgS2
System.out.println("Chave: "+rsa.getsPublicKey());
msg.enviaMsgS2(outputStream, rsa.getsPublicKey());

//envia msgS3
String chaveSecreta = rsa.cifraDados(aes.getsKeySpec());
System.out.println("Chave simétrica = "+aes.getsKeySpec());
msg.enviaMsgS3(outputStream, chaveSecreta);

// recebe msgC2
msgC2 = Autenticacao.C2.parseDelimitedFrom(inputStream);
System.out.println("Id Cliente = "+aes.decifraDados(msgC2.getIdCli()));

//envia msgS4
String idServidor = aes.cifraDados("123456789");
msg.enviaMsgS4(outputStream, idServidor);

// recebe msgC3
msgC3 = Autenticacao.C3.parseDelimitedFrom(inputStream);
System.out.println("Code = "+aes.decifraDados(msgC3.getCode()));

//envia msgS5
String idSessao = aes.cifraDados("1");
msg.enviaMsgS5(outputStream, idSessao);

// recebe msgC4
msgC4 = Autenticacao.C4.parseDelimitedFrom(inputStream);
System.out.println("Fim = "+msgC4.getFim());

//envia msgS6
msg.enviaMsgS6(outputStream, Boolean.TRUE);
. . .
```

Cliente - AES

```
. . .
S1 msgS1;
S2 msgS2;
S3 msgS3;
S4 msgS4;
S5 msgS5;
S6 msgS6;

Mensagens msg = new Mensagens();
RSAJ2ME rsa;
AESJ2ME aes;

// envia msgC1
msg.enviaMsgC1(outputStream);

// recebe msgS1
msgS1 = S1.parseDelimitedFrom(inputStream);
System.out.println("Id Sessão = "+msgS1.getIdSess());

// recebe msgS2
msgS2 = S2.parseDelimitedFrom(inputStream);
rsa = new RSAJ2ME(msgS2.getChPub());

// recebe msgS3
msgS3 = S3.parseDelimitedFrom(inputStream);
aes = new AESJ2ME(rsa.decifraDados(msgS3.getChSim()));

// envia msgC2
msg.enviaMsgC2(outputStream, aes.cifraDados("1234567890987654"));

//recebe msgS3
msgS4 = S4.parseDelimitedFrom(inputStream);
System.out.println("IdServidor = "+aes.decifraDados(msgS4.getIdSer()));

// envia msgC3
msg.enviaMsgC3(outputStream, aes.cifraDados("786hgff"));

// recebe msgS5
msgS5 = S5.parseDelimitedFrom(inputStream);
System.out.println("Id Sessão: "+aes.decifraDados(msgS5.getIdSess()));

// envia msgC4
msg.enviaMsgC4(outputStream, true);

// recebe msgS6
msgS6 = S6.parseDelimitedFrom(inputStream);
System.out.println("Fim = "+msgS6.getFim());
. . .
```

4.4.4 Resultados / Conclusão

A Tabela 3 apresenta os tempos de execução do processo de autenticação quando se combina o *RSA* com os algoritmos simétricos (*DES* e *AES*), para diferentes tamanhos de chave e diferentes meios de comunicação (intranet e 3G).

Tabela 3 – Resultados RSA/DES e RSA/AES

Algoritmo	**Chave (bits) – *RSA***	**Chave (bits) – *DES/AES***	**Tempo de execução**	
			Intranet	**Internet – Placa TMN 3G**
RSA* e *DES	512	56	1 seg	1 seg
	1024		1 seg	1 seg
RSA* e *AES	512	128	1 seg	1 seg
	1024		1 seg	2 seg

Os resultados obtidos evidenciam que é possível implementar, utilizando a estrutura do *protocol buffer*, mecanismos de segurança que operam em várias meios de comunicação. As diferenças, que teoricamente poderiam existir nos tempos de execução do processo de autenticação com as diferentes combinações, não são visíveis na prática. A explicação para este aspecto poderá estar relacionada com o facto de existir, nos testes realizados, um único cliente ligado ao servidor. A gestão por parte deste é relativamente simples, possibilitando respostas rápidas aos vários pedidos realizados pelo cliente. No entanto, fica evidenciado que o mecanismo de segurança proposto funciona e que com uma implementação optimizada do servidor é possível aproximar os tempos de execução dos tempos apresentados.

Face as estes resultados, a implementação de um mecanismo de segurança com a combinação *RSA* (1024) e *AES* (128) afigura-se com sendo a melhor opção, o que confere ao projecto em estudo um mecanismo de autenticação robusto e funcional.

5. Estrutura de Mensagens

O conceito de mensagem é utilizado para a comunicação verbal ou escrita. Todas as partes envolventes têm "de falar a mesma linguagem" de forma a interpretar o conteúdo das mensagens. A definição de um conjunto de regras claras e precisas, permite a estruturação de mensagens interpretáveis pelas várias partes envolvidas. No campo das novas tecnologias, o conceito de mensagem permite o envio e recepção através de serviços digitais de comunicação. Esta técnica de transmissão de informação é utilizada com frequência em sistemas de informação (Porto Editora, 2010).

No projecto em estudo, optou-se por utilizar a comunicação baseada em mensagens, tanto na comunicação entre a aplicação cliente e o *middleware* como entre este e o servidor HL7.

As regras de estruturação de mensagens a trocar entre as aplicações são definidas mais adiante neste capítulo. Relativamente à troca de mensagens entre o *middleware* e o servidor HL7, utilizou-se naturalmente a norma HL7. A gestão das mensagens é da responsabilidade do *middleware*.

5.1 Mensagens Cliente – Middleware

O *middleware* é responsável por garantir a troca de informação entre a aplicação cliente e o servidor HL7. Para que tal fosse possível, houve a necessidade de definir regras claras e objectivas, possibilitando desta forma a interpretação da informação trocada.

Na estruturação da informação a transmitir optou-se por implementar o *protocol buffer*, que utiliza o conceito de mensagem para troca de dados. As mensagens são definidas em estruturas padronizadas e simples de implementar, conforme mostra o seguinte exemplo:

```
Pessoa {
        required string name = 1;
        required int32 id = 2;
        optional string email = 3;
}
```

Os dados a transmitir pela aplicação cliente estão sujeitos às regras definidas para a estruturação de mensagens no *protocol buffer*. Esta implementação restringe-se unicamente à definição de regras na construção do conteúdo dos dados a transmitir, ou seja, na construção do esqueleto de dados tendo em conta o tipo de informação a enviar.

Assumiu-se que os dados a transferir entre sistemas devem ser simples, devendo estes, no entanto, apresentar alguma ambiguidade, de forma a garantir que determinado paciente não seja identificado.

As mensagens subjacentes ao processo de criação de ambientes seguros, que garantem segurança na transmissão de informação entre as várias partes integrantes, são também aqui normalizadas.

A identificação das mensagens trocadas pelos vários componentes é necessária para determinar a tarefa a realizar com a informação recebida. Nesse sentido, implementou-se um mecanismo que consiste na incorporação, em cada mensagem, de um identificador que tem como objectivo associar a mensagem à respectiva tarefa a realizar pelo componente receptor (Tabela 4).

Tabela 4 – Tipo de Mensagem

Tipo de Mensagem	Valor
Autenticação	1
PedidoObjectivo	2
PedidoSenhaEspera	3
RespostaObjectiva	4
RespostaSenhaEspera	5

O tratamento de erros é outro aspecto para o qual a aplicação deve estar preparada. Neste sentido, e à semelhança do identificador do tipo de mensagem, implementou-se um mecanismo que permite associar o erro a um identificador (Tabela 5).

Tabela 5 – Lista de Erros

Identificador	Descrição
1	Erro_1
2	Erro_2
…	…
n	Erro_n

Todas as estruturas de mensagem definidas neste capítulo são meramente exemplificativas, podendo ser reajustadas às funcionalidades pretendidas.

5.1.1 Mensagens de Autenticação

Neste capítulo, definiu-se unicamente o tipo e o nome dos campos que constituem as mensagens trocadas no processo de autenticação, utilizando a estrutura de dados proposta pelo *protocol buffer*. A descrição sucinta de todo o procedimento de autenticação está documento no Capítulo 4. A Figura 15 representa as mensagens trocadas ao longo do processo de autenticação de um cliente.

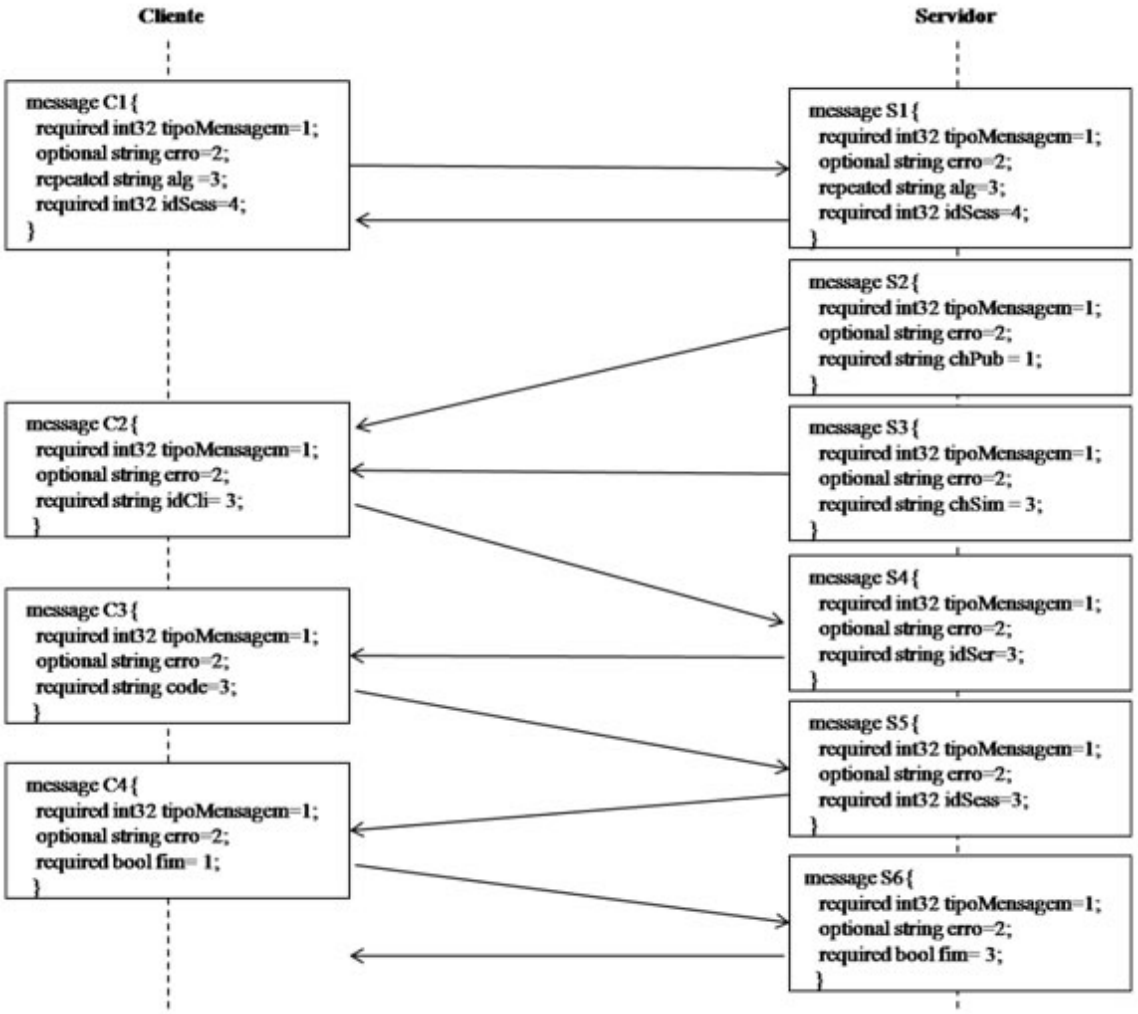

Figura 15 - Estrutura de mensagens trocadas ao longo do processo de autenticação de um cliente

5.1.2 Mensagens Pedido/Resposta

Após a conclusão do processo de autenticação, é necessário definir um conjunto de regras para a estruturação de mensagens a trocar (pedidos/respostas). No projecto em estudo, assumiu-se que as mensagens a partilhar entre sistemas devem conter dados simples, como por exemplo, o valor máximo

e mínimo da tensão arterial. O principal objectivo da definição de estruturas de dados, idêntica à norma HL7, é normalizar o formato das mensagens.

As mensagens são classificadas em dois grupos, sendo que cada um dos grupos é constituído por dois tipos de mensagens:

Pedido	**Resposta**
• Pedido Objectivo	• Resposta Objectiva
• Pedido Senha de Espera	• Resposta Senha de Espera

O Pedido Objectivo e a Resposta Objectiva são, respectivamente, a informação solicitada por parte do cliente ao servidor e a resposta à informação solicitada. O Pedido/Resposta Senha de Espera é um mecanismo adoptado que permite associar uma senha de espera a um pedido ou resposta, garantindo que a aplicação cliente não fica eternamente bloqueada à espera de uma resposta por parte do servidor. Periodicamente, a aplicação cliente utiliza a senha de espera para verificar se existe resposta ao seu pedido. Este tema será aprofundado no Capítulo 6.

Pedido Objectivo

O Pedido Objectivo é responsável por transportar a informação referente a um determinado pedido realizado por um cliente. A Tabela 6 identifica os campos que constituem a mensagem do tipo Pedido Objectivo.

Tabela 6 – Campos que constituem a estrutura da mensagem do tipo Pedido Objectivo

Campo	Descrição
tipoMensagem	Identificador da mensagem enviada
erro	Identificador do erro ocorrido
idSess	Identificador da sessão do cliente
idCliente	Identificador do cliente
idPaciente	Identificador do paciente
idObservacao	Identificador da observação a analisar
numRegistos	Número de registos a observar
prioridade	Prioridade do pedido. Por defeito é 1

- **idObservacao –** representa o identificador de determinada observação, definida numa lista de observações. Esta lista de observações, passível de alteração, contém o conjunto de observações possíveis de presenciar (Tabela 7).

Tabela 7 - Lista de Observações

Identificador Observação	Observação
1	Obs_1
2	Obs_2
…	…
n	Obs_n

- **numRegistos –** define, para determinado idObservacao, as últimas n amostragens a consultar;
- **prioridade** – define a prioridade do pedido (Tabela 8).

Tabela 8 – Lista de Prioridades

Prioridade	Nível
1	Normal
2	Alto
3	Urgente

A estrutura seguinte representa a implementação da mensagem do tipo Pedido Objectivo no *protocol buffer*.

```
PedidoObjectivo{
        required int32 tipoMensagem=1;
         optional string erro=2;
         optional int32 idSess=3;
         required string idCliente=4;
         required int32 idPaciente=5;
         required int32 idObservacao=6;
         required int32 numRegistos=7;
         optional int32 Prioridade=8; [default = 1]
 }
```

Pedido Senha de Espera

A mensagem Pedido Senha de Espera consiste no envio de uma senha de espera, previamente gerada e enviada ao cliente por parte do *middleware*, que identifica o Pedido Objectivo efectuado anteriormente. No *protocol buffer* a mensagem Pedido Senha de Espera apresenta a seguinte estrutura:

```
PedidoSenhaEspera{
        required int32 tipoMensagem=1;
        optional string erro=2;
        option int32 idSess=3;
        required string idCliente=4;
        required long senhaEspera= 5;
}
```

Resposta Objectiva

A Resposta Objectivo tem como função transportar a resposta de um determinado pedido. A Tabela 9 identifica os campos que constituem a estrutura de uma mensagem do tipo Resposta Objectivo.

Tabela 9 – Campos que constituem a estrutura da mensagem do tipo Resposta Objectiva

Campo	Descrição
tipoMensagem	Identificador da mensagem enviada
erro	Identificador do erro ocorrido
idPaciente	Identificador do paciente
idObservacao	Identificador da observação a analisar
valRegisto	Valores do registo
resposta	Indicador da resposta a um pedido

- **IdObservação –** apresenta o mesmo significado descrito para a estrutura da mensagem do tipo Pedido Objectivo;
- **valRegisto -** dependendo do Pedido Objectivo, o campo **valRegisto** pode assumir mais do que um valor, como por exemplo, a tensão arterial é representada habitualmente por um valor máximo e um valor mínimo. Este campo é transportado numa lista de tamanho **numRegistos,** onde cada entrada na lista contém o(s) valor(es) de cada registo, separados pelo carácter **' / ',** caso o registo seja constituído por mais do que um valor.

A mensagem do tipo Resposta Objectivo apresenta a seguinte implementação na estrutura do *protocol buffer*:

```
RespostaObjectiva{
        required int32 tipoMensagem=1;
         optional string erro=2;
         required int32 idPaciente=3;
         required int32 idObservacao=4;
         repeated string  valRegistos= 5;
         required bool resposta= 6;
}
```

Resposta Senha de Espera

A mensagem Resposta Senha de Espera consiste no envio, por parte do *middleware*, de uma senha de espera única que identifica um Pedido Objectivo realizado anteriormente. A implementação desta mensagem no *protocol buffer* apresenta a seguinte estrutura:

```
RespostaSenhaEspera{
        required int32 tipoMensagem=1;
        optional string erro=2;
        required long senhaEspera=2;
}
```

5.2 HL7

A *Health Level Seven*[4] é uma organização, fundada em 1987, acreditada pela *American National Standards Institute*[5] (ANSI) para o desenvolvimento de normas para a área da saúde. Esta organização providencia uma norma para a partilha, gestão e integração de dados de suporte aos serviços de cuidados de saúde. O termo "*Level 7*" refere-se ao nível mais elevado do modelo de comunicação OSI da *International Organization for Standardization*[6] (ISO) (Sui-hui *and* Ni-Ni, 2005).

A especificação de uma norma para a área da saúde acarreta um conjunto de vantagens, nomeadamente:

- Existência de um sistema aberto;
- Constituição de uma norma única, universal de integração, partilha e recuperação de informação electrónica na saúde;
- Criação de uma rede global de registos clínicos.

5.2.1 Norma HL7

Conforme referido anteriormente, a norma HL7 providencia uma sintaxe para a representação e partilha de grandes quantidades de informação médica, permitindo definir a forma como a informação é estruturada, não interpretando, no entanto, o conteúdo das mensagens. A Figura 16 ilustra o modo de funcionamento da norma HL7.

[4] http://www.hl7.org/
[5] http://www.ansi.org/
[6] http://www.iso.org

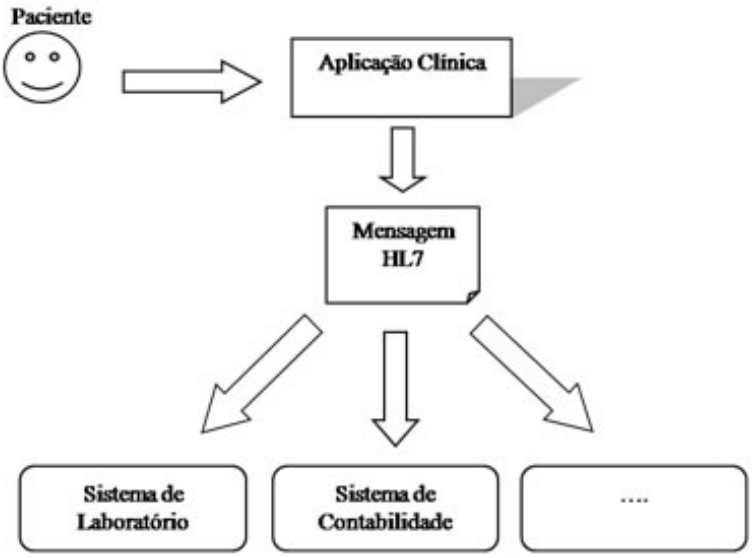

Figura 16 – Funcionamento do HL7

A utilização desta norma permite que as aplicações clínicas comuniquem entre si, independentemente da arquitectura de dados, da plataforma tecnológica e da linguagem de programação. Actualmente, a norma HL7 apresenta 3 versões, as quais serão descritas a seguida.

HL7v1

Apresentada em 1987, esta versão define o formato de representação das mensagens. Esta implementação, muito limitada em termos de funcionalidade, evoluiu rapidamente para a versão 2.

HL7v2.x

Em 1988 surgiu a versão 2, que após sofrer algumas actualizações, foi acreditada em 1994 pela ANSI, passando a ser uma norma com utilização prática na área da saúde. Esta versão é actualmente a mais utilizada no universo da saúde.

A utilização de diferentes versões dentro da HL7v2 não é um aspecto limitativo uma vez que existe compatibilidade entre as várias versões.

As mensagens HL7v2.x representam informação com origens diferentes, como resultados e observações clínicas, troca de dados de pacientes, pedidos dos serviços clínicos (farmácia, enfermaria, …), sendo representadas em texto ASCII. A Figura 17 mostra o exemplo de uma mensagem HL7v2.3.1.

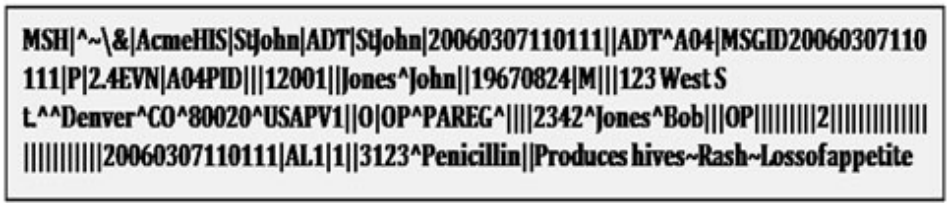

Figura 17 – Exemplo de uma mensagem HL 7 v.2.3.1 (Neotool, 2007)

A norma HL7v2.x apresenta algumas limitações que a comunidade HL7 pretendeu resolver, designadamente (Neotool, 2007):

- Ausência de um modelo de dados;
- Ausência de regras aplicacionais e de utilização;
- Redução do grau de flexibilidade e opções;
- Fraca interoperabilidade com novas tecnologias (*XML*, *WEB*).

HL7v3

As limitações apresentadas pela HL7v2 levaram a que um subconjunto da comunidade HL7 propusesse a criação de uma nova versão, a HL7v3, definindo para esta, um conjunto de objectivos (Neotool, 2007):

- **Internacionalização** – Capacidade de qualquer organização HL7 utilizar a norma HL7v3 para resolver as necessidades das variantes locais;
- **Modelo de dados consistente** – A HL7v3 necessita de definir um modelo de dados, usado por aplicações HL7, de forma a garantir a consistência dos dados;
- **Norma precisa** – A HL7v3 necessita de aproveitar a informação aprendida com todas as versões HL7v2.x, criando uma norma que contém todos os dados necessário e que seja pouco vaga e pouco flexível;
- **Nova norma –** Quando a comunidade começou a definir a HL7v3, foi decidido que a versão 3 seria incompatível com a versão 2 por inúmeras razões. Primeiramente, se a versão 3 fosse compatível com a versão 2, esta nova versão estaria condicionada pelos problemas herdados da versão 2. Qualquer tentativa de reabilitar dados explícitos ou modelos aplicacionais da versão 2 seria muito difícil. Finalmente, a norma necessita de espaço para respirar, podendo mudar radicalmente melhorando desta forma a qualidade das *interfaces* clínicas.

No âmbito deste trabalho, foi utilizado a norma HL7v3 como parte integrante na comunicação de mensagens entre a camada *middleware* e servidor HL7.

5.2.2 Plataforma de Desenvolvimento de Mensagens HL7v3

A Plataforma de Desenvolvimento de Mensagens *(Messsage Development Framework)* define o processo de desenvolvimento de mensagens HL7v3 através da criação de modelos e gráficos distribuídos por várias etapas. Cada etapa representa uma tarefa específica e está associada a um modelo ou um grupo de modelos.

Etapa I – Requisitos de Mensagem

Os requisitos de desenvolvimento começam por definir o objectivo do novo projecto, proporcionando uma descrição, através do desenvolvimento do Modelo de Casos de Uso (*Use Case Model),* dos processos de actividade.

O Modelo de Casos de Uso é construído utilizando diagramas de casos de uso que capturam as especificações do projecto, permitindo a definição completa dos requisitos que o conjunto de mensagens projectadas deve suportar. O Modelo de Casos de Uso fornece a base que garante, ao longo das etapas seguintes, qualidade no processo de desenvolvimento de mensagens.

Etapa II – Conteúdos de Mensagem

A análise estrutural focaliza-se no Modelo de Informação (*Information Model*), que define os dados transportados pelas mensagens e analisa as transições de estados de *classes* e subconjunto de *classes* que desempenham um papel preponderante no desenvolvimento de mensagens. O Modelo de Informação é construído utilizando diagramas de *classes* e diagramas de transição de estados. A criação do Modelo de Informação proporciona uma definição consistente e clara do conteúdo de uma mensagem ou de um grupo de mensagens relacionadas.

O Modelo de Referência de Informação (*Reference Information Model)* representa o modelo de informação de conteúdos para a norma HL7. Em Anexo, encontra-se um exemplar de um Modelo de Referência de Informação. A utilização do Modelo de Referência de Informação, como fonte de conteúdo de mensagens, proporciona uma maior eficiência no processo de desenvolvimento de mensagens.

O desenvolvimento de um modelo de transição de estados para subconjuntos de *classes,* cria uma ligação entre o trabalho desenvolvido nesta etapa e a investigação do comportamento das mensagens a desenvolver na etapa seguinte.

Etapa III – Comportamento da Mensagem

A análise comportamental focaliza-se no Modelo de Interacção (*Interaction Model*), que define a especificação de interacções (fluxo de informação) necessárias ao suporte dos requisitos funcionais. As interacções identificam os eventos que originam a partilha de informação e indicam também em que situações essas mensagens são necessárias.

O Modelo de Interacção é construído utilizando as definições de funções aplicacionais de *classes*, uma grelha tabular de interacções e diagramas de sequência que descrevem as interacções específicas (fluxo de informação) e as funções aplicacionais necessárias ao suporte dos requisitos de mensagem. As interacções providenciam um padrão para o fluxo de mensagens entre aplicações que utilizam a norma HL7.

Etapa IV – Especificação da Mensagem

A especificação envolve o uso de modelos criados durante as etapas anteriores, de forma a gerar especificações de mensagens, que definem com exactidão as mensagens HL7.

O Modelo de Informação de Mensagens *(Message Information Model)* é um subconjunto do Modelo de Referência de Informação, que contém os dados relevantes para cada mensagem ou grupo de mensagens. A Descrição Hierárquica de Mensagens (*Hierarchical Message Description)* providencia uma representação tabular que mostra os atributos que fazem parte de cada mensagem e define, para os componentes de mensagem, a presença ou ausência de eventos. A Especificação Tecnológica de Implementação (*Implementation Technology Specification)* define com exactidão a representação do conteúdo da Descrição Hierárquica de Mensagens num formato de mensagem específico, como o *XML* (Beeler *et al*., 1999).

A Figura 18 mostra os modelos que são criados ou modificados ao longo do processo de desenvolvimento de mensagens HL7.

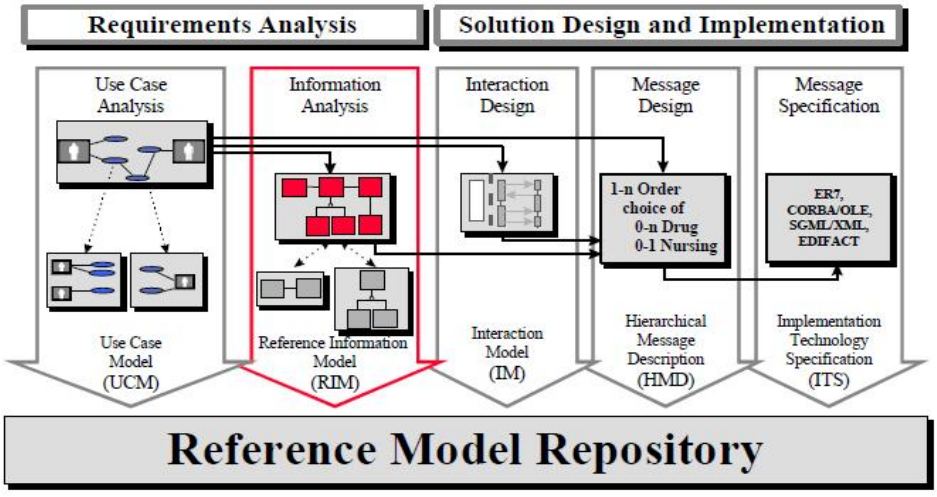

Figura 18 – Plataforma de Desenvolvimento de Mensagens HL7v3 (Beeler et al., 1999)

6. Arquitectura do Sistema de Saúde

O sistema de saúde proposto é composto por um conjunto de componentes (hardware e software) interligados entre si, permitindo a transmissão de informação. Neste capítulo, analisou-se os vários componentes, concretamente a aplicação cliente e a aplicação *middleware*.

6.1 Serviços e Interfaces

Os vários componentes que integram qualquer arquitectura ou sistema têm tarefas associadas. As tarefas podem ser entendidas como serviços que quando invocados produzem uma resposta a um determinado pedido, sendo o modo de execução dos mesmos um processo transparente para o utilizador.

Actualmente, os serviços são definidos por uma ou mais *interfaces*, dependendo da complexidade do mesmo (Alexandre, 1978). A *interface* descreve a forma de interacção entre quem solicita o serviço e quem o executa, sendo definida segundo um conjunto de regras previamente padronizadas através do conceito de *Interface Description Language* (IDL).

O *protocol buffer*, através do *Proto Definition file* (*.proto*), define estruturas de dados genéricas, que servirão de suporte à execução de serviços.

6.2 Mecanismo de Comunicação

A existência de vários componentes pressupõe a definição de mecanismos de comunicação. Neste projecto, optou-se pela utilização da tecnologia *protocol buffer* que define a estrutura de dados a transmitir, assentando o sistema num mecanismo de comunicação orientado a mensagens.

A comunicação pode ser realizada de forma assíncrona ou síncrona. Na comunicação assíncrona (Figura 19), o cliente envia o pedido para o servidor e continua a executar outras tarefas. O servidor, depois de receber e executar a operação solicitada, envia o resultado produzido ao cliente ou armazena-o para ser consultado posteriormente. Na comunicação síncrona (Figura 20) o cliente realiza o pedido e fica bloqueado à espera de resposta.

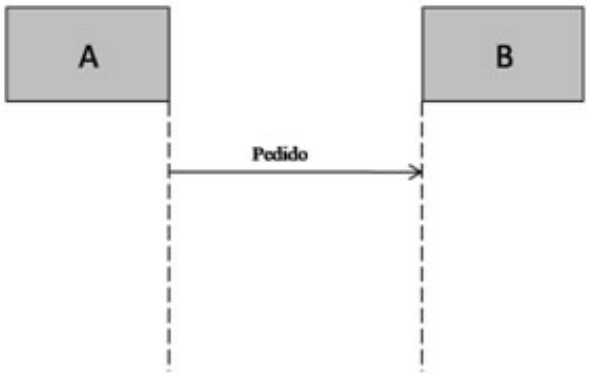

Figura 19 – Comunicação Assíncrona

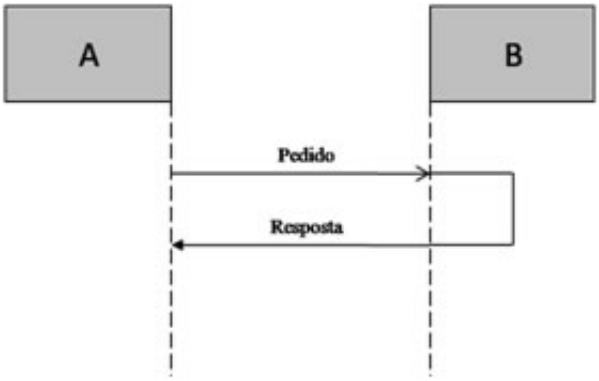

Figura 20 - Comunicação Síncrona

6.3 Padrões

O termo padrão foi introduzido pelo Arquitecto Christopher Alexandre, que propõe a ideia original de construir catálogos de padrões para a arquitectura. Um padrão descreve um problema que ocorre inúmeras vezes num determinado contexto e a respectiva solução, de modo que essa solução possa ser utilizada sistematicamente em situações distintas (Alexandre, 1978).

Este conceito de padrão foi importado para as novas tecnologias, classificando-se segundo o tipo de problema, ligado ao desenvolvimento de software, ou solução que partilham uma estrutura semelhante. Actualmente, os dois principais padrões são, os padrões de arquitectura e os padrões de desenho (Choppy *et al.*, 2006).

6.3.1 Padrões de Arquitectura

Padrões de arquitectura são padrões de alto nível que proporcionam uma visão geral de todo o sistema. Normalmente, representam os principais componentes, as propriedades externas visíveis, as principais funcionalidades e a relação entre os componentes (Nickull *et al.*, 2007). Neste projecto, optou-se por dar mais ênfase à arquitectura multi-níveis (Krakowiak, 2009).

Arquitectura baseada em Camadas

A decomposição em camadas é uma ferramenta organizacional poderosa e muito utilizada na concepção de sistemas complexos pois permite construir uma visão simplificada de todo o sistema. As camadas são desenvolvidas de forma independentemente, tornando o processo de desenvolvimento mais fácil, bem como o processo de actualização e de manutenção. A comunicação entre camadas é realizada com recursos a *interfaces*, onde cada uma possui um conjunto de funções, agrupadas em livrarias, que fornecem acesso a serviços (Krakowiak, 2009).

Arquitectura *Multitier*

O sistema tradicional cliente/servidor é constituído por dois módulos, o cliente e o servidor. O servidor, habitualmente uma base de dados, é responsável pelo armazenamento de informação, enquanto o cliente é constituído por uma interface gráfica responsável pela parte do processamento. Estas arquitecturas são caracterizadas por serem orientadas a dados. As desvantagens da sua utilização residem no processo de gestão de versões e na distribuição de actualizações de *software* cliente (Shan *and* Hua, 2006).

A arquitectura multi-camadas (*multitier)* surgiu para resolver o problema inerente ao modelo cliente/servidor. Em vez de assentar em duas camadas, cliente e servidor, introduziu-se uma terceira camada aplicacional que se localiza entre o cliente e o servidor, usualmente uma base de dados. Esta camada é responsável por garantir a interoperabilidade entre os vários intervenientes. A camada cliente passa a ter unicamente funções de interacção com o utilizador, enquanto a gestão da base de dados é da responsabilidade da camada servidor (Krakowiak, 2009). Neste tipo de arquitectura as três camadas estão localizadas em máquinas separadas, conforme mostra a Figura 21.

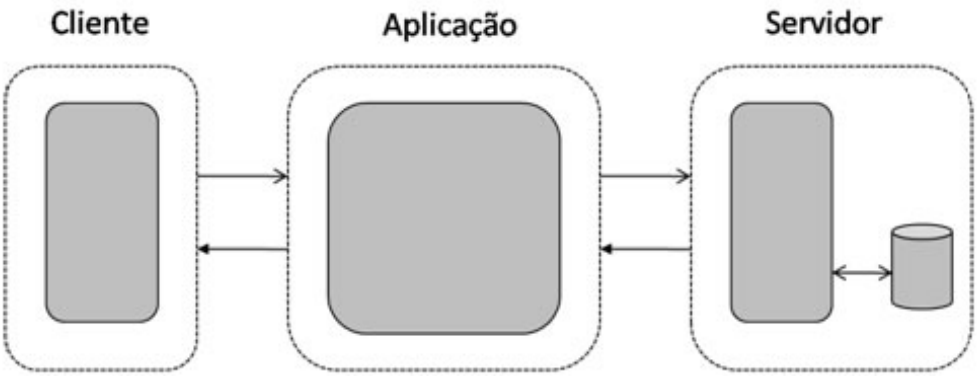

Figura 21 – Arquitectura *multitier* (Krakowiak, 2009)

6.3.2 Padrões de Desenho

Padrões de desenho providenciam soluções viáveis para implementar subsistemas ou componentes integrantes de um sistema mais complexo e definem a relação entre eles. Descrevem, também, estruturas, devidamente testadas, de componentes de comunicação que em determinado contexto resolvem o problema apresentado (Nickull *et al.*, 2007). Os padrões de desenho são classificados, segundo Gamma *et al.* (1995), em três categorias, de acordo com a Tabela 10.

Tabela 10 – Classificação de padrões de desenho segundo GoF

Categorias		
Criação	**Estruturais**	**Comportamentais**
Abstract Factory	Adapter	Interpreter
Builder	Bridge	Template Method
Factory Method	Composite	Chain of Responsability
Prototype	Facade	Command
Singleton	Flyweight	Iterator
	Proxy	Mediator
		Memento
		Observer
		State
		Strategy
		Visitor

6.4 Padrões para troca de Mensagens

Em sistemas do tipo cliente/servidor a troca de dados é habitualmente realizada através de mensagens. A definição de uma sequência lógica de envio e recepção de mensagens é um processo que deve ser definido e interpretado pelos vários intervenientes. Este padrão representa um conjunto de normas que providenciam sequências para a troca de mensagens. Os modelos mais conhecidos são, o *One-Way*, o *Publish/Subscribe* e o *Request/Reply*.

One-Way

Este padrão, muito simples, caracteriza-se pelo envio de uma mensagem num determinado sentido (Figura 22).

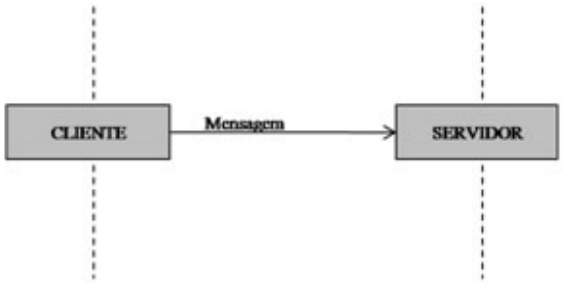

Figura 22 – Padrão O*ne-Way*

Publish/Subscribe

Publish/Subscribe é actualmente uma tecnologia muito utilizado na difusão de informação (Corsado *et al.*, 2006). Este tipo de padrão caracteriza-se pela existência de um publicador (*Publisher*) e um subscritor *(subscriber).* O subscritor subscreve um determinado tópico ou conteúdo e recebe todos os eventos publicados relacionados com a subscrição (Pesonen *et al.*, 2006). Cada participante pode ser, simultaneamente, publicador e subscritor (Figura 23).

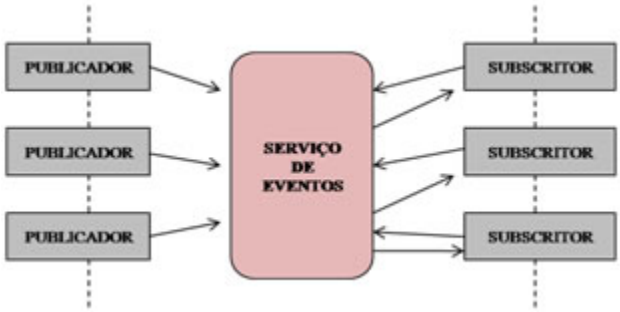

Figura 23 – Padrão *Publish/Subscribe*

Request/Reply

Este padrão caracteriza-se pela obrigatoriedade de resposta a um pedido (Figura 24). O cliente envia para o servidor o pedido e fica bloqueado à espera de uma resposta. O protocolo de transporte *HTTP* implementa este tipo padrão (Yu *et al.*, 2006).

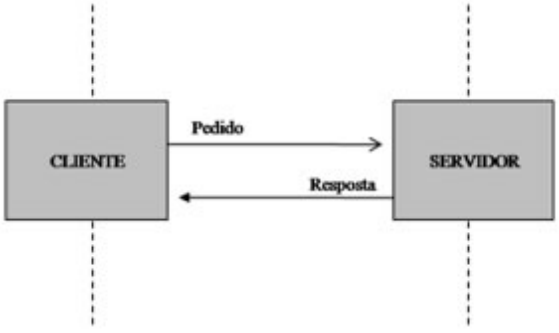

Figura 24 – Padrão *Request/Replay*

6.5 Message Oriented Middleware

Message Oriented Middleware (MOM) assenta num sistema distribuído de comunicação síncrono ou assíncrono e persistente de troca de mensagens. Os componentes elementares que constituem este sistema são, as aplicações clientes, as mensagens e o provedor (*provider*), que inclui ferramentas administrativas.

Num sistema com estas características, o cliente invoca a função de envio de uma mensagem para um destinatário conhecido do provedor. O provedor providencia serviços que garantem a entrega da mensagem. Depois do envio da mensagem, o cliente continua a trabalhar confiante que o provedor retém a mensagem até esta ser recuperada por um determinado cliente (Sun Microsystems, 2007c). A Figura 25 mostra o modo de funcionamento de um sistema MOM.

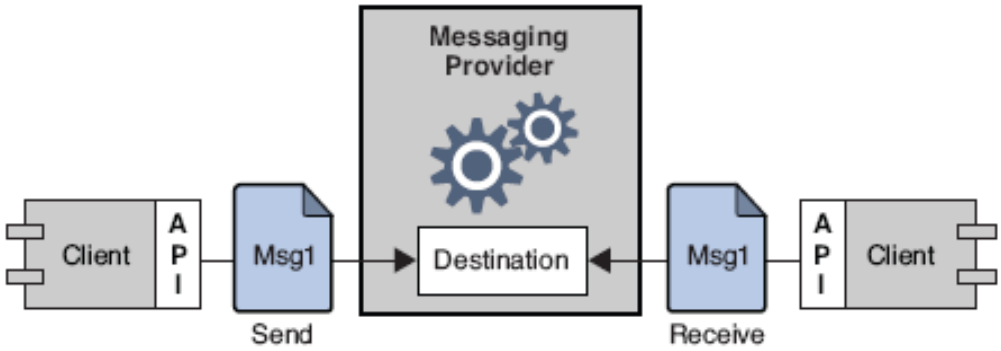

Figura 25 – Sistema MOM (Sun Microsystems, 2007c)

6.6 Arquitectura de Componentes

A análise de sistemas é um processo exigente, onde a probabilidade da ocorrência de erros é grande. A subdivisão do sistema em vários componentes permite tratar, de forma independente, cada um deles, podendo o resultado ser utilizado como parâmetro de entrada num outro componente. Esta abordagem proporciona a elevação para um nível de abstracção de todo o sistema, aumentando assim a compreensão e concepção do mesmo. A dependência que possa existir entre os componentes, é um problema para o qual é necessário ter particular atenção aquando da análise e implementação de um sistema com estas características.

O sistema de saúde projectado, caracteriza-se por apresentar uma arquitectura *multitier*, baseado no modelo de distribuição orientado a mensagens. A transmissão de mensagens é realizada de forma assíncrona, num sistema com propriedades semelhantes ao modelo *Publish/Subscribe*. A tecnologia utilizada na estruturação de mensagens foi o *protocol buffer*, pela sua simplicidade e rapidez de transmissão.

O sistema encontra-se agrupado em três camadas principais, a aplicação cliente, a aplicação *middleware* e o servidor HL7. Cada camada é um componente constituído por sub-componentes interligados entre si e com tarefas específicas atribuídas.

6.6.1 Aplicação Cliente

A aplicação cliente é a ferramenta utilizada para solicitar a informação desejada e apresentá-la ao utilizador. A arquitectura da aplicação, apresentada na Figura 26, é composta por um conjunto de componentes, sendo que a cada um está associada uma determinada tarefa. O processo de implementação da arquitectura foi condicionado pelo facto da aplicação cliente funcionar em dispositivos móveis, que apresentam algumas limitações de recursos.

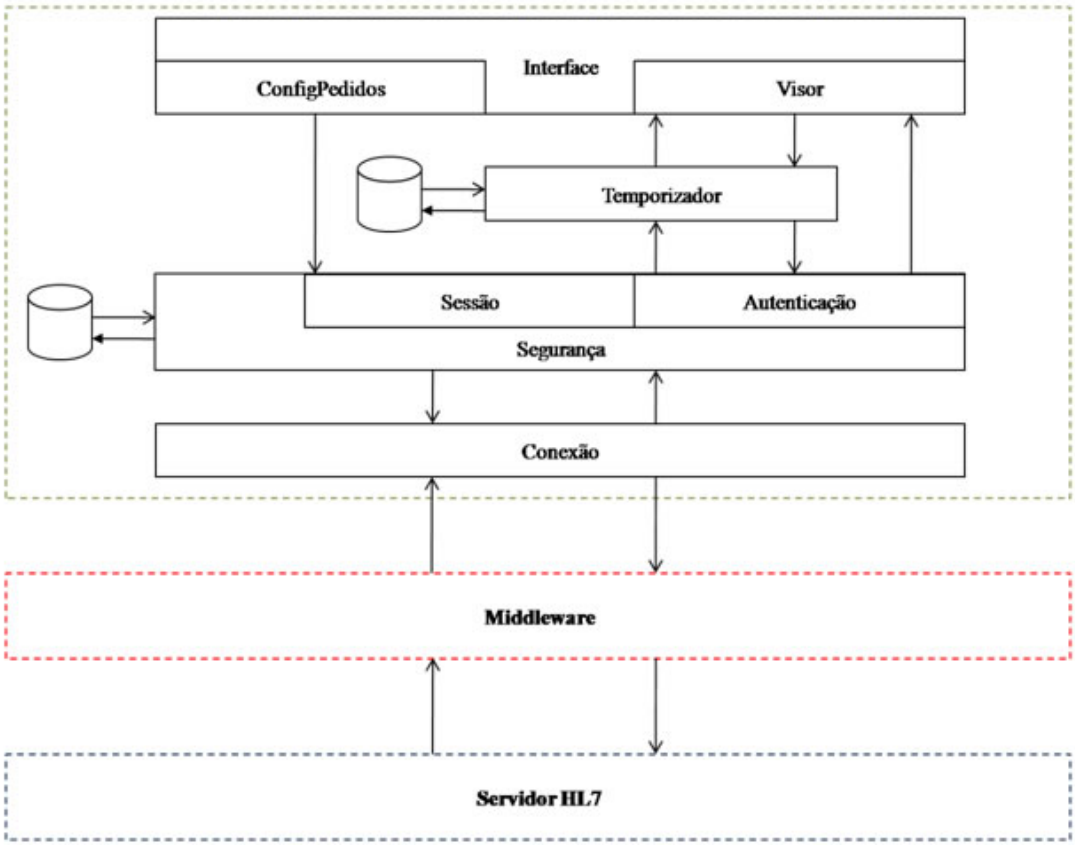

Figura 26 – Arquitectura da aplicação cliente

A seguir, é apresentada a descrição das funcionalidades de cada um dos componentes constituintes da aplicação cliente.

Interface

Este componente é responsável por estabelecer a comunicação com o utilizador, sendo constituído por outros dois componentes fundamentais, designadamente o ConfigPedidos e Visor. O

primeiro é responsável por possibilitar a configuração de pedidos e a funcionalidade de disponibilizar visualmente a informação é da responsabilidade do componente Visor.

Temporizador

A função principal do temporizador é permitir que periodicamente seja solicitado ao *middleware*, através de senha de espera, resposta aos vários pedidos anteriormente configurados. Toda a gestão de pedidos é realizada por este componente.

Segurança

A segurança é um aspecto importante quando a informação a transmitir é sensível. A incorporação deste componente visa equipar o sistema com funções de segurança, nomeadamente cifrar e decifrar dados, estabelecer o processo de autenticação ou de sessão, ou seja, implementar o protocolo de segurança descrito no Capítulo 4.

Conexão

O estabelecimento de ligações, entre os diferentes componentes, é a base para a existência de uma arquitectura de camadas, sendo este requisito preenchido com a introdução do componente Conexão.

6.6.1.1 Modo de Funcionamento

O modelo proposto assenta na possibilidade dos utilizadores poderem configurar um conjunto de pedidos que são enviados, após a execução com êxito do processo de autenticação, para a aplicação *middleware.* Esta aplicação ao receber os pedidos, gera um número aleatório único, que funciona como senha de espera e que é posteriormente enviado ao cliente. Cada senha de espera está relacionada com um pedido, ou seja, por cada pedido efectuado por determinado cliente, o *middleware* atribui um número aleatório único. A aplicação cliente ao receber a senha de espera, guarda-a, verificando periodicamente, através do envio da senha de espera para o *middleware,* se existe resposta aos seus pedidos. Caso a resposta seja positiva, a aplicação cliente mostra o resultado no visor, ou guarda a informação para posterior visualização, sendo a senha de espera removida. A senha de espera mantém-se activa, apenas no caso da resposta ser negativa. A Figura 27 mostra o modo de funcionamento da aplicação cliente.

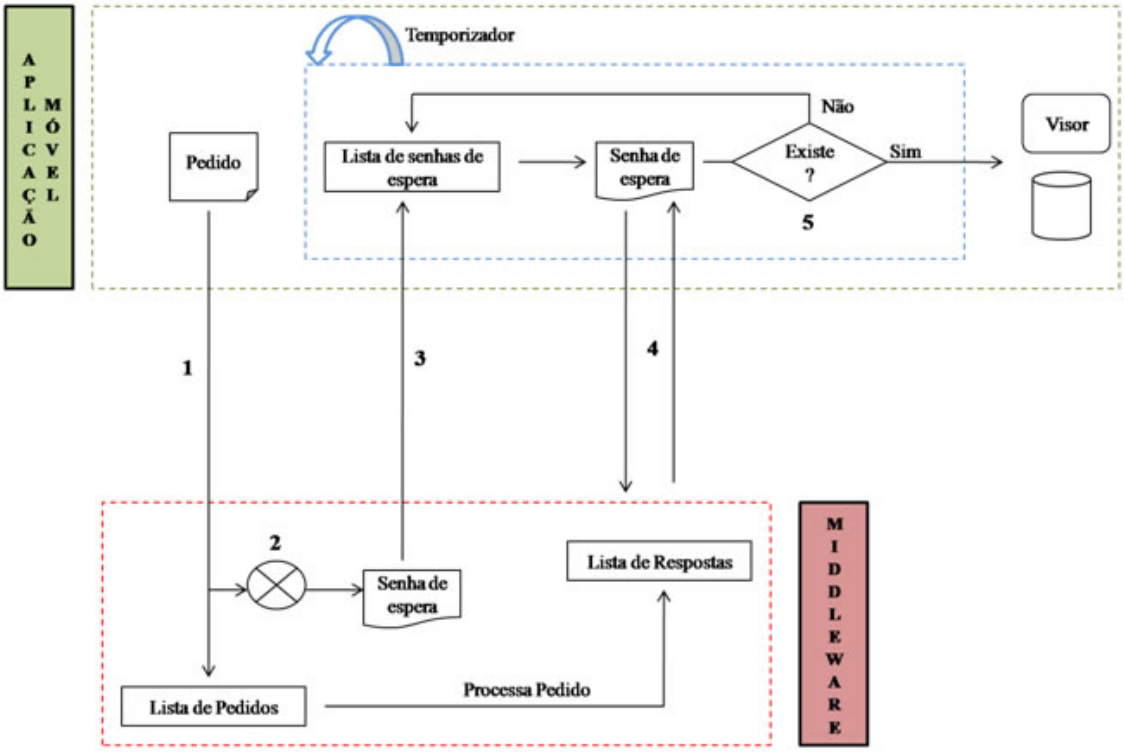

Figura 27 – Funcionamento da aplicação cliente

O funcionamento da aplicação cliente caracteriza-se por operar de forma assíncrona. Esta abordagem permite que a aplicação cliente não fique eternamente bloqueada por uma resposta do *middleware*, que por alguma falha técnica pode nunca acontecer.

6.6.2 Aplicação Middleware

A arquitectura do sistema *middleware* adoptada na implementação deste projecto (Figura 28), à semelhança da aplicação cliente, é constituída por um conjunto de componentes com determinadas tarefas atribuídas. Caracteriza-se por apresentar funcionalidades idênticas ao componente provedor, que constitui o núcleo central do modelo MOM, garantindo a entrega da informação solicitado pela aplicação cliente.

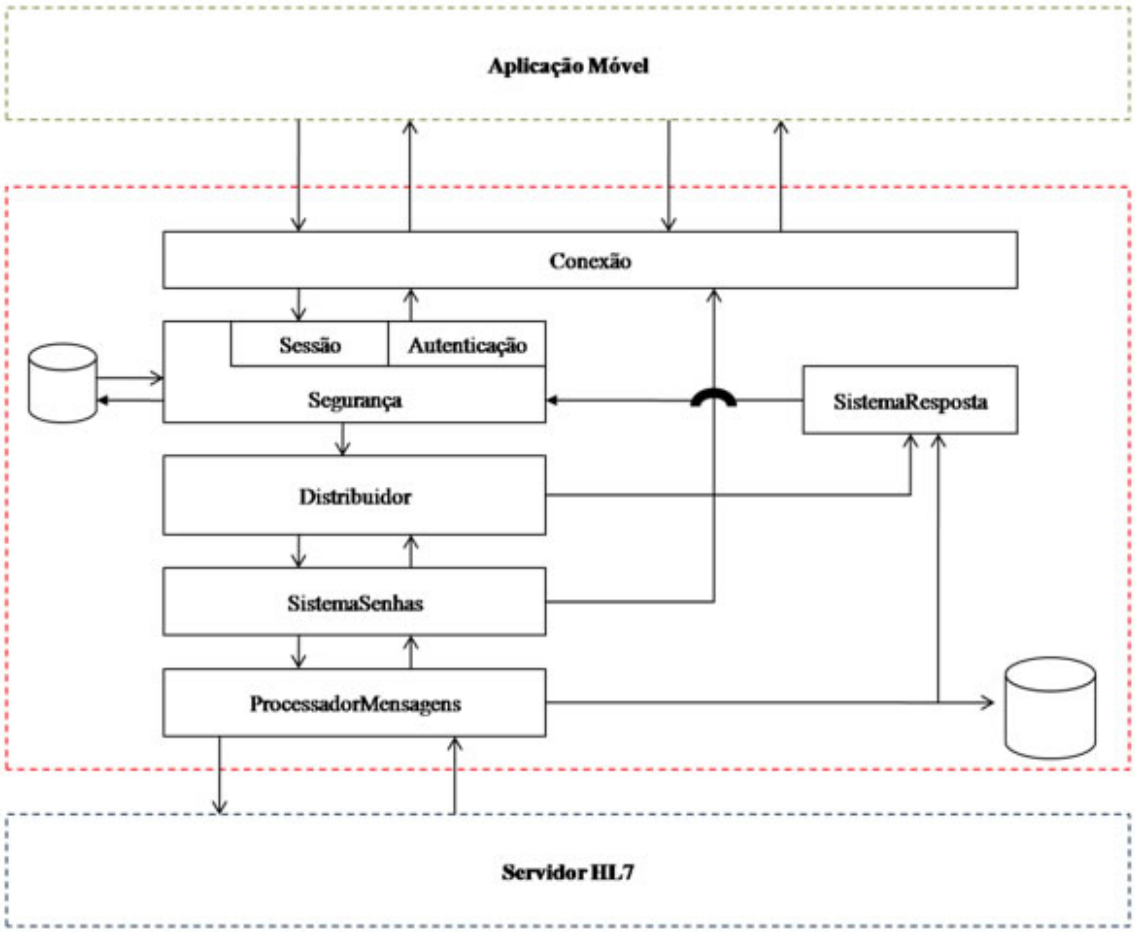

Figura 28 – Arquitectura da aplicação *middleware*

A seguir, é apresentada a descrição das funcionalidades de cada um dos componentes constituintes da aplicação *middleware*.

Conexão e Segurança

A função dos componentes Conexão e Segurança é a mesma descrita na aplicação cliente.

Distribuidor

A função do componente Distribuidor é verificar e encaminhar as mensagens, consoante o seu tipo, para o componente responsável pelo tratamento do pedido.

SistemaSenhas

Este componente tem a função de gerar um número aleatório único, capaz de identificar um pedido realizado pela aplicação cliente. A gestão dos números únicos é da responsabilidade deste componente.

ProcessadorMensagens

Este componente é responsável por construir, a partir do pedido do cliente e segundo a norma HL7v3, mensagens que serão posteriormente enviadas ao servidor HL7. A acção contrária, ou seja, construir mensagens cliente a partir de mensagens enviadas pelo servidor HL7, é também da responsabilidade deste componente.

SistemaResposta

O componente SistemaResposta tem como objectivo disponibilizar respostas (Resposta Objectiva) aos pedidos solicitados.

6.6.2.1 Modo de Funcionamento

O *middleware* é a camada responsável pelo tratamento das mensagens enviadas pela aplicação cliente. A descrição a seguir apresentada, assume que o processo de autenticação e de criação ou reutilização de sessão foi previamente efectuado. As mensagens são dirigidas para o componente Distribuidor que as verifica e as encaminha, consoante o tipo, para o componente responsável pelo tratamento. Caso seja uma mensagem do tipo Pedido Objectivo, é gerado um número aleatório, que funciona como senha de espera, e é enviado para a aplicação cliente. Esta senha de espera é acoplada à mensagem de forma a relacionar o pedido com a resposta posteriormente gerada. A mensagem Pedido Objectivo é convertida numa mensagem HL7 que é enviada para o servidor HL7. Este processa o pedido e responde com uma mensagem HL7 que é transformada, pelo componente ProcessadorMensagens, numa mensagem do tipo Resposta Objectiva. A resposta ao pedido é armazenada, para posteriormente ser encaminhada para a aplicação cliente. Caso a mensagem seja do tipo Pedido Senha de Espera, esta é reencaminhada para o componente SistemaResposta, que verifica se existe resposta para essa senha de espera. Se existir, é enviado à aplicação cliente a mensagem Resposta Objectiva. A Figura 29 mostra o modo de funcionamento da aplicação *middleware*.

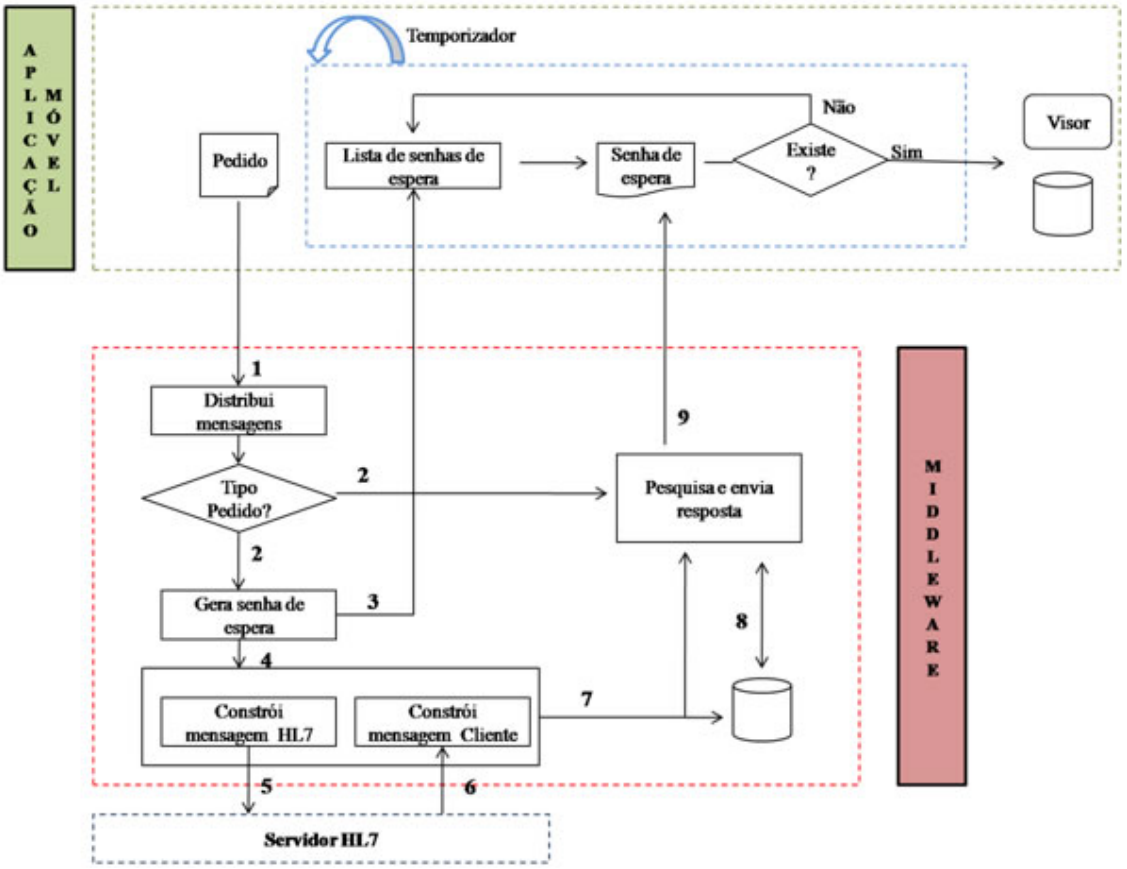

Figura 29 - Funcionamento da aplicação *middleware*

6.7 Thread Polling Pattern

Actualmente, o modelo cliente/servidor é implementado para fornecer serviços que são consumidos por clientes. Os serviços estão alojados num servidor que executa de forma concorrente, ou seja, por cada solicitação cria um processo ou *thread* que tem como principal objectivo fornecer resposta ao pedido. As *threads* são um mecanismo muito utilizado para implementar o conceito de concorrência, no entanto, é bastante penoso para o servidor que tem de criar por cada pedido uma *thread*. O tempo desperdiçado na criação de *threads* é elevado, aumentando ao longo do período de execução do servidor. Para solucionar este problema, foi desenvolvido o conceito de fila de *threads* (*thread pool*), que contém um número pré definido de *threads* que são criadas aquando da execução inicial do servidor. Para cada pedido, é retirado da fila uma *thread* que processa o pedido, a qual é reposta novamente na fila de *threads* depois de terminar a execução. Este modelo é estático e apresenta algumas limitações. Para baixas quantidades de pedidos, poderão ser alocados recursos de sistema que nunca irão ser utilizados, por outro lado, se o número de pedidos for muito elevado, existe o risco de não satisfazer todos os pedidos por falta de recursos.

Neste projecto procurou-se implementar um modelo de fila de *threads* flexível, capaz de adaptar o número de *threads* à quantidade de pedidos existentes. O conceito de flexibilidade é atingido

com a introdução de um mecanismo capaz de prever o número de *threads* necessárias numa próxima actualização da fila de *threads*. Um modelo que preenche os requisitos pretendidos, encontra-se documentado no artigo de Kang *et al.* (2008). A Figura 30 mostra o modo de funcionamento pretendido para a gestão da fila de *threads*. O modelo foi desenhado para melhorar o tempo de resposta do servidor e proporcionar uma gestão mais eficiente dos recursos do sistema.

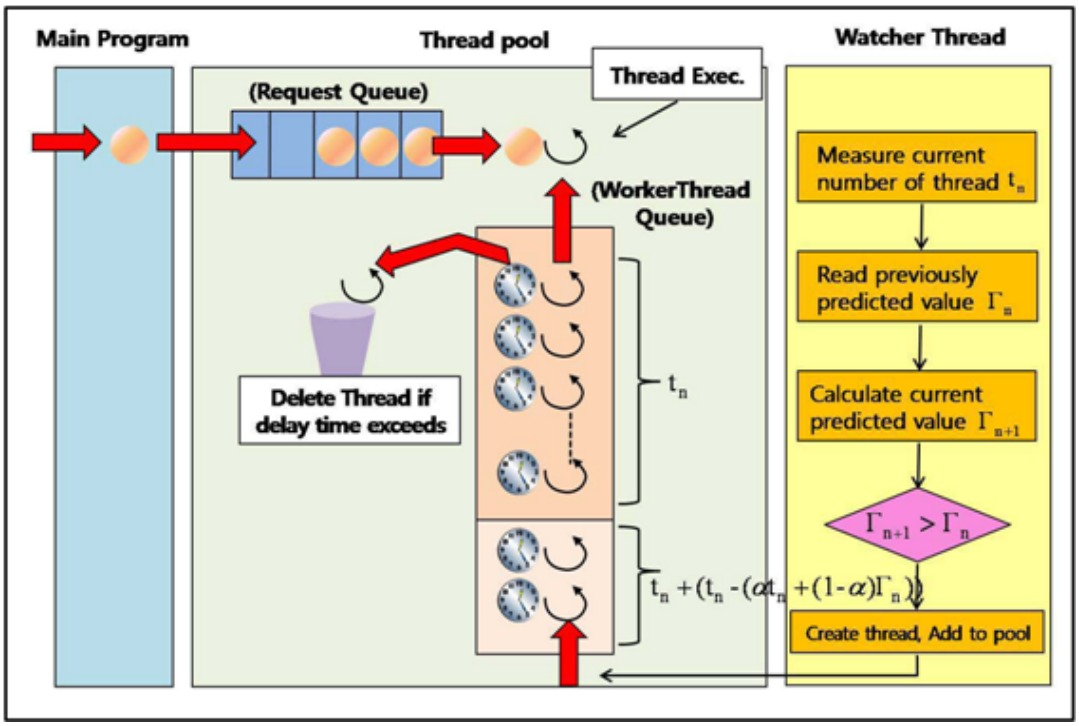

Figura 30 - Modelo de funcionamento da fila de *threads* (Kang *et al.*, 2008)

6.8 Protótipo

O principal objectivo do protótipo é demonstrar, através de um caso prático, que a arquitectura descrita anteriormente tem aplicabilidade prática, oferecendo aos seus utilizadores inúmeras vantagens. Nesta implementação, o caso prático abordado visa a monitorização, por parte de um profissional de saúde, dos níveis de glicose ou tensão arterial de um determinado paciente. O protótipo é composto por dois componentes essenciais, a aplicação cliente e o *middleware*.

6.8.1 Aplicação Cliente

A aplicação cliente desenvolvida reflecte o modo de funcionamento demonstrado na Figura 27, sendo apresentado na Figura 31 o diagrama de casos de uso.

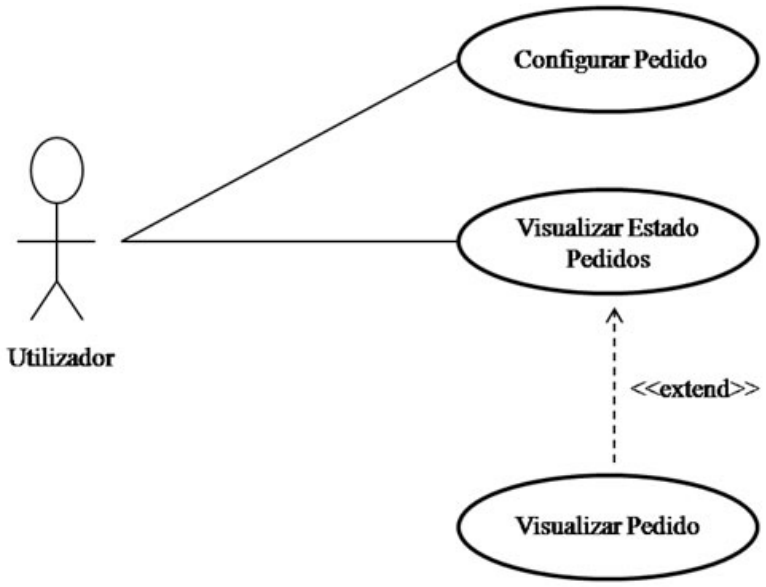

Figura 31 – Diagrama de casos de uso

Visão geral de implementação

A aplicação cliente foi desenvolvida em *J2ME*, utilizando para o efeito a plataforma de desenvolvimento *NetBeans* 6.7.1 e testada com o emulador, *Java Plataform Micro Edition SDK 3.0,* que simula a execução da aplicação num dispositivo móvel. A Figura 32 mostra o modelo de desenho da aplicação cliente, projectado com a ajuda da ferramenta *Flow Designer*.

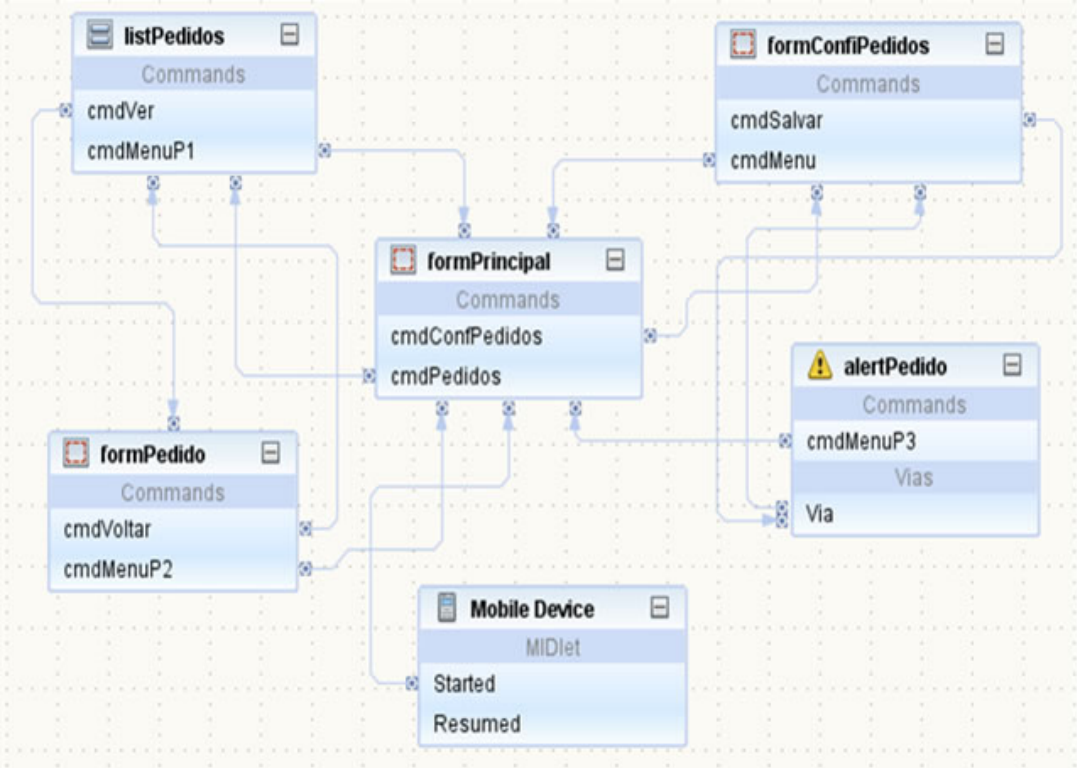

Figura 32 – Modelo de desenho da aplicação cliente

A seguir, descreveu-se e ilustrou-se, através de figuras, o modelo de desenho da aplicação cliente:

- **formPrincipal -** Ecrã Principal;
- **formConfigPedidos –** Configurar um pedido;
- **listPedidos –** Lista de todos os pedidos e o estado em que se encontram (Aguarda ou Respondido);
- **formPedido –** Lista resposta a determinado pedido;
- **alertPedido –** Mostra mensagem de erro.

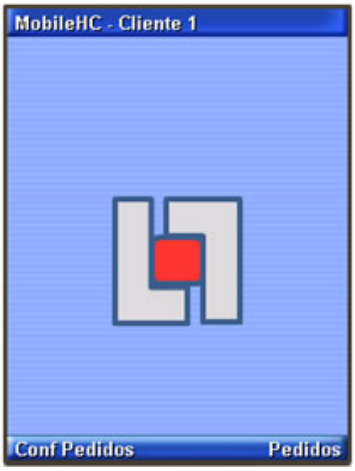

Figura 33 - formPrincipal

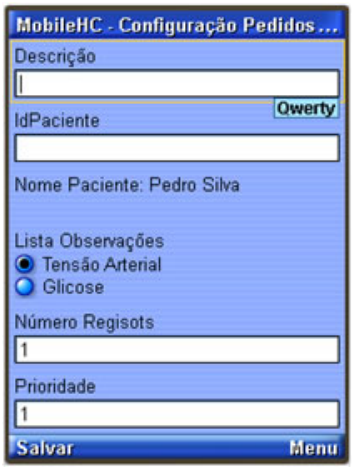

Figura 33 – formConfigPedidos

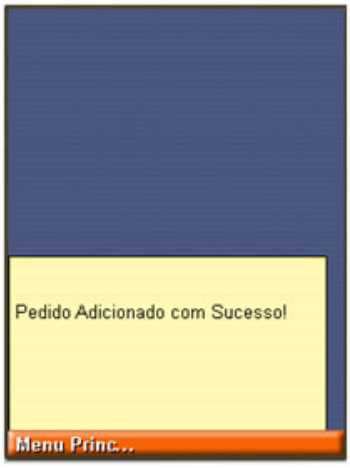

Figura 34 – alertPedido

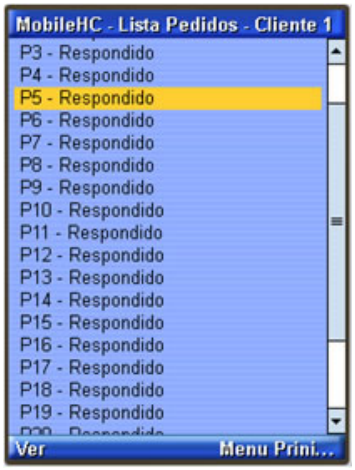

Figura 35 – listPedidos

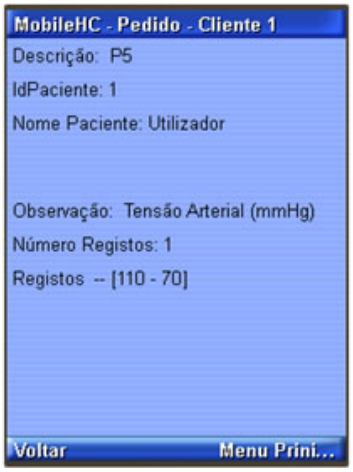

Figura 36 – formPedido

Ao longo de todo o processo de desenvolvimento foram ainda considerados alguns aspectos importantes, nomeadamente questões relacionadas com a interacção utilizador – aplicação.

Caso prático

O cliente começa por configurar os pedidos que são enviados ao *middleware*, o qual os processa e responde com a respectiva senha de espera. O processo de resposta ao pedido solicitado é um processo assíncrono. Após a configuração dos pedidos, o utilizador tem a possibilidade de visualizar o estado em que esses se encontram e caso tenham obtido resposta, o utilizador pode consultá-la. A Figura 38 mostra as funcionalidades que a aplicação cliente disponibiliza ao seu utilizador.

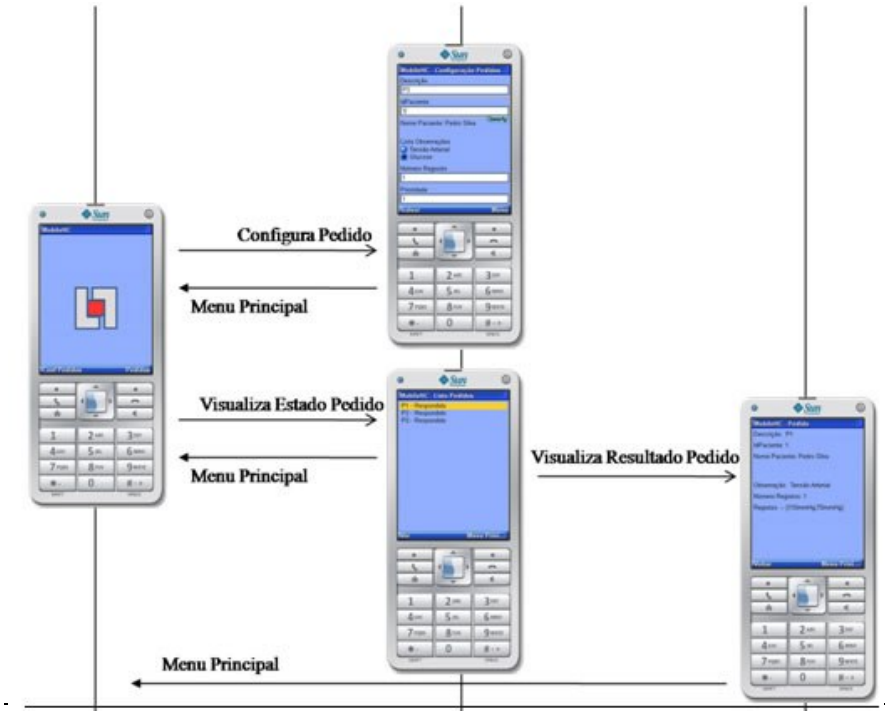

Figura 37 – Funcionalidades práticas da aplicação cliente

6.8.2 Aplicação Middleware

Os serviços desempenhados pelo *middleware* devem apresentar duas características fundamentais, disponibilidade e fiabilidade. No sentido de medir o grau de fiabilidade dos serviços prestados, foi projectado um cenário de teste (Figura 39) onde dois clientes móveis simulam o envio de um conjunto de pedidos (10), em intervalos de tempo de 1 segundo, durante 5 iterações. Com base no número de pedidos não respondidos é possível aferir o grau de fiabilidade do sistema *middleware*.

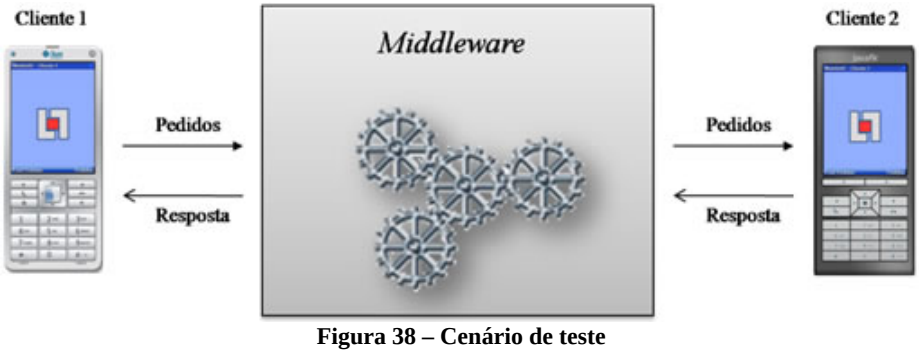

Figura 38 – Cenário de teste

Após a execução do cenário de teste retratado anteriormente, é possível concluir que o grau de fiabilidade da aplicação *middleware* é bastante razoável, tendo em conta que todos os pedidos efectuados obtiveram resposta. Convém, no entanto, salientar que os testes foram desenvolvidos num ambiente controlado, o que poderá condicionar os resultados finais. O cenário de testes ideal seria um

ambiente real, que por motivos logísticos não foi possível implementar. No entanto, os resultados obtidos são encorajadores e positivos para trabalhos futuros.

Visão geral de implementação

A implementação da aplicação *middleware* obedeceu à arquitectura e modo de funcionamento definidos nas Figuras 28 e 29, respectivamente. O seguinte diagrama de classes mostra a estrutura de classes implementada no projecto em estudo.

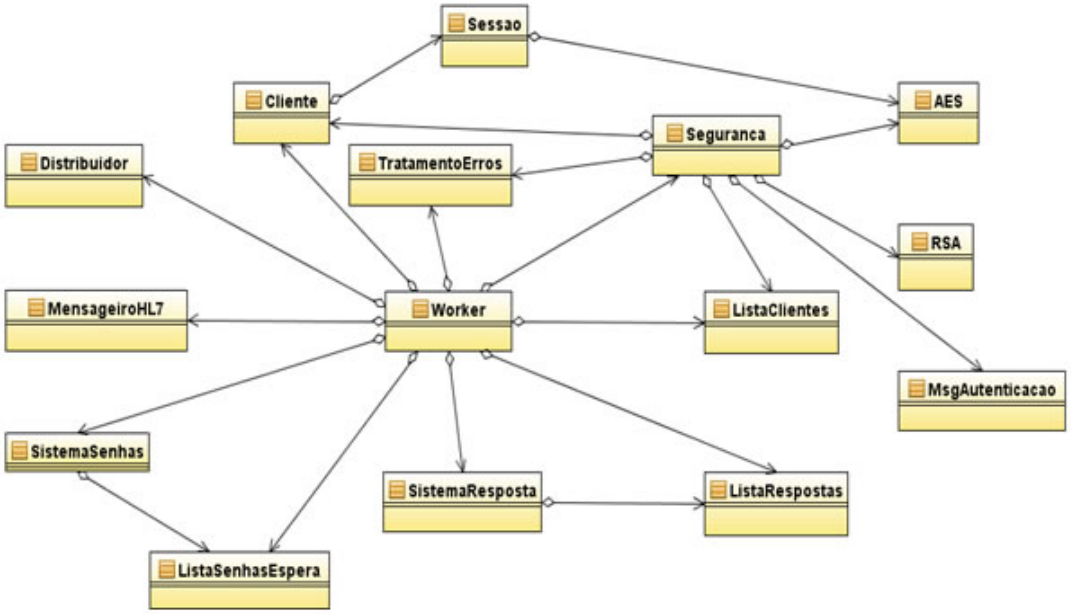

Figura 39 – Diagrama de classes

Implementação da fila de *threads*

Relativamente ao padrão de fila de *threads* implementado para este sistema (Figura 30), o objecto *Thread pool* foi implementado com recurso à *classe Executor*, presente no *package java.util.concurrent***,** conforme mostra a seguinte estrutura de código.

```
…
// Cria uma fila de threads variável
Executor exec = Executors.newCachedThreadPool();

while(true){
      clientSocket = serSocket.accept();

      // Processa um novo pedido com uma thread retirada da fila de threads
      exec.execute(new Worker(clientSocket, listaClientes, listaSenhasEspera, listaRespostas));
}
…
```

O objecto *Watcher Thread* não foi implementado devido à necessidade de definição de um algoritmo que se adapte ao sistema de saúde proposto. O desenvolvimento de um algoritmo com estas características é um processo demoroso que envolve a recolha de um conjunto de dados relevantes, como por exemplo, a média de pedidos efectuados durante um certo período de tempo, não sendo esse o objectivo definido para esta dissertação.

7. Conclusões

Esta dissertação descreve um trabalho realizado sobre o acesso, a partir de dispositivos móveis, a registos médicos de pacientes previamente monitorizados, utilizando como camada intermédia um sistema *middleware*. O desenvolvimento de um sistema com estas características, envolve um conjunto de aspectos que foram amplamente analisados e documentados.

Foram estudados e testados dois protocolos de comunicação diferentes, o *protocol buffer* e os *web services*, optando-se pela utilização do primeiro neste projecto. Ambas as tecnologias apresentam vantagens e desvantagens, no entanto, os testes realizados indicam que o *protocol buffer* apresenta tempos de execução mais reduzidos, quando comparado com os *web services*. Outro aspecto importante que influenciou na opção de utilizar o *protocol buffer*, prende-se com o facto de a sua estrutura de mensagem permitir, de forma simples e intuitiva, a construção de mensagens que são partilhadas entre os vários componentes que constituem o sistema projectado. No entanto, a implementação do *protocol buffer* poderá levantar alguns problemas, nomeadamente quando existem restrições de segurança, como por exemplo, *firewalls*. Estas barreiras de segurança são minimizadas nos *web services*, em virtude de estes utilizarem como protocolo de transporte o *HTTP*. Este problema poderá, num trabalho futuro, ser ultrapassado com a combinação do *protocol buffer* com o *HTTP*.

A segurança foi outro aspecto amplamente estudado, visando conferir ao sistema um mecanismo de segurança capaz de responder aos critérios de segurança pretendidos. Foram analisados três sistemas de cifras diferentes, nomeadamente cifra simétrica (chave secreta), cifra assimétrica (chave pública) e cifra híbrida, que combina as vantagens das duas cifras anteriores. As vantagens advêm do facto do sistema de cifra simétrica apresentar tempos de execução mais reduzidos e taxas de consumo de recursos mais baixas, quando comparado com o sistema de cifra assimétrica. Este apresenta-se, no entanto, como uma solução que confere segurança na distribuição da chave secreta. Juntamente com as respectivas chaves criptográficas, foram ainda analisados e testados os tempos de execução de algoritmos criptográficos existentes, tendo-se focado com particular atenção o *DES* e o *AES*, para cifras simétricas, e o *RSA*, para cifras assimétricas. De salientar que a combinação dos algoritmos *AES* e *RSA*, de 128 e 1024 bits, respectivamente, traduz em termos práticos e com bons tempos de execução, os critérios de segurança pretendidos. Foi também projectado um protocolo de segurança que descreve, através de mensagens, o processo de autenticação de um determinado cliente.

Na procura de normalizar as mensagens partilhadas, foram projectadas estruturas de mensagens para os vários cenários possíveis, nomeadamente envio de pedidos e respostas. A norma HL7 foi introduzida de forma a padronizar as mensagens trocadas com sistemas que implementam HL7.

A arquitectura do sistema proposto, foi implementada tendo em consideração a utilização de padrões. O sistema divide-se em três camadas fundamentais: aplicação cliente, aplicação *middleware* e servidor HL7, caracterizando-se por ser orientado a mensagens e com uma comunicação assíncrona. Projectou-se, para a camada cliente e camada *middleware*, o desenho de uma possível arquitectura baseada em componentes. A camada do servidor HL7 foi parcialmente desenvolvida, podendo ser aprofundada num trabalho futuro.

A forma de tratamento de pedidos, por parte do *middleware*, foi outro aspecto estudado, optando-se por implementar um padrão de fila de *threads* flexível. Esta flexibilidade é conseguida com a introdução de um algoritmo que calcula a estimativa de pedidos futuros para posteriormente reajustar a fila de *threads.* Este algoritmo, por não ser considerado um dos objectivos principais do projecto em estudo, não foi implementado, podendo ser desenvolvido num trabalho futuro.

Após todo o processo de análise e documentação das arquitecturas, foi implementado um protótipo, com recurso à linguagem de programação *Java*, que simulou a recolha e visualização de sinais vitais de um paciente monitorizado. Os resultados de carga realizados revelam que todos os pedidos enviados para a camada *middlware* obtiveram resposta, o que permite concluir que o sistema é fiável. A aplicação cliente teve o comportamento esperado, permitindo obter e disponibilizar, ao seu utilizador, os dados solicitados.

8. Referências bibliográficas

ADNAN, Syed Farid Syed; HASHIM, Habibah. “Medical mobile to web services application.” In 5th International Colloquium on Signal Processing & Its Applications, 2009. CSPA 2009, pp. 124-129. IEEE Conferences, 2009.

Aerotel. *Mobile-CliniQ*. 2010. Disponível em http://www.aerotel.com/en/images/products/Marketing/aerotel-catalog-2010_-_e-cliniq.pdf. Acedido em 20 Junho 2010.

ANDERSON, Ross. “Security Engineering: A Guide to Building Dependable Distributed Systems.” Wiley, 2008. 129 - 183.

ALEXANDRE, Christopher. “A Pattern Language.” Oxford Press, Reino Unido, 1978.

Apache Software Foundation. *Web Services – Axis*. 2010. Disponível em http://ws.apache.org/axis/java/install.html. Acedido em 7 Fevereiro 2010.

BEELER, George; HUFF, Stan; RISHEL, Wesley; SHAHIR, Abdul-Malik; WALKER, Mead; MEAD, Charlie; SCHADOW, Gunther. Health Level Seven Internacional. *HL7 version 3: Development Framework*, p. 16(1-5)-18(1-7). 1999. Disponível em http://www.hl7.org/implement/standards/ReferenceMaterials/mdf99.pdf. Acedido em 26 Dezembro 2009.

BOUKERCHE, Azzedine; REN, Yonglin. “A secure mobile healthcare system using trust-based multicast scheme.” *In IEEE Journal on Selected Areas in Communications, pp. 387-399.* IEEE JOURNALS, 2009.

CHAN, V.; RAY, P.; PARAMESWARAN, N.. “Mobile e-Health monitoring: an agent-based approach.” *In Communications, IET, pp. 223 - 230.* IET, 2008.

CHOPPY, C.; HATEBUR , D.; HEISEL, M.. “Architectural patterns for problem frames.” *Software, IEE Proceedings, pp. 198-208.* IET, 2006.

CORSADO, Angelo; QUERZONI, Leonardo; SCRIPIONI, Sirio. *Quality of Service in Publish/Subscribe Middleware*, 2006. IOS Press.

CRK, Igor; ALBINALI, Fahd; GNIADY, Chris; HARTMAN, John. "Understanding energy consumption of sensor enabled applications on mobile phones." *In Annual International Conference of the IEEE on Engineering in Medicine and Biology Society, 2009. EMBC 2009, pp. 6885-6888* . IEEE CONFERENCES, 2009.

DENG, Liqiong; POOLE, Marshall. "Learning Through Telemedicine Networks." *In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003.* IEEE CONFERENCES, 2003.

GADDAH, Abdulbaset; KUNZ, Thomas. "A Survey of Middleware Paradigms for Mobile Computing." Carleton University Systems and Computing Engineering Technical Report SCE-03-16, 2003.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software*, 1ª Edição, Addison-Wesley, 1995.

Google Code. *Developer Guide - Protocol Buffer Basics Google Code*. 2010. http://code.google.com/intl/pt-PT/apis/protocolbuffers/docs/overview.html. Acedido em 14 Fevereiro 2010.

Google Code. *protobuf-javame - Java ME implementation of Google Protocol Buffers.* 2009. Disponível em http://code.google.com/p/protobuf-javame/. Acedido em 14 Fevereiro 2010.

Google Code. *Protocol Buffer Basics: Java*. 2010. http://code.google.com/intl/pt-PT/apis/protocolbuffers/docs/javatutorial.html. Acedido em 6 Fevereiro 2010.

Google Open Source Blog. *Protocol Buffers: Google´s Data Interchange Format.* 2008. Disponível em http://google-opensource.blogspot.com/2008/07/protocol-buffers-googles-data.html. Acedido em 14 Maio 2010.

HL7. *HL7 Reference Information Model*. 2009. Disponível em http://www.hl7.org/library/data-model/RIM/modelpage_mem.htm. Acedido em 26 Dezembro 2009.

KANG, DongHyun; HAN, Saeyoung; YOO, SeoHee; PARK, Sungyong. “Prediction-Based Dynamic Thread Pool Scheme for Efficient Resource Usage.” *In IEEE 8th International Conference on Computer and Information Technology Workshops, 2008. CIT Workshops 2008, pp. 159-164.* IEEE CONFERENCES, 2008.

KNUDSEN, Jonathan. *Parsing XML in J2ME.* 2002. Disponível em http://developers.sun.com/mobility/midp/articles/parsingxml/. Acedido em 13 Fevereiro 2010.

KRAKOWIAK, Sacha. *Middleware Architecture with Patterns and Frameworks*. 2009. Disponível em http://sardes.inrialpes.fr/~krakowia/MW-Book/Chapters/Preface/preface.html. Acedido em 01 Maio 2010.

Microsoft. *Descrição de handshake por Secure Sockets Layer (SSL)*. 2007. Disponível em http://support.microsoft.com/kb/257591. Acedidos em 11 Março 2010.

MobiMed. *Mobile Medical solution for Hospitals.* 2006. Disponível em http://www.mobimed.org/index.html. Acedido em 17 Maio 2010.

Neotool. *The HL7 Evolution: Comparing HL7 Version 2 to Version 3, Including a History of Version 2*. 2007 Disponível em http://www.corepointhealth.com/sites/default/files/whitepapers/hl7-v2-v3-evolution.pdf. Acedido em 24 Dezembro 2009.

NICKULL, Duane; REITMAN, Laurel; WARD, James; WILBER, Jack. *Service Oriented Architecture (SOA) and Specialized Messaging Patterns.* Adobe Systems Incorporated, 2007.

PESONEN, Lauri; EYERS, David; BACON, Jean. “A Capability-Based Access Control Architecture for Multi-Domain Publish/Subscribe Systems.” *In International Symposium on Applications and the Internet, 2006. SAINT 2006, pp 7-228.* IEEE CONFERENCE, 2006.

Porto Editora. *Definição de Mensagem no Dicionário da Língua Portuguesa da Porto Editora*. 2010. Disponível em http://www.infopedia.pt/lingua-portuguesa/Mensagem. Acedido em 04 Abril 2010.

RFC5246. *The TLS Protocol version 1.2*. 2008. Disponível em http://tools.ietf.org/html/rfc5246. Acedidos em 11 Março 2010.

RSA Laboratories. *Chapter 3 Techniques in Cryptography*. 2010. RSA Laboratories. Disponível em http://www.rsa.com/rsalabs/node.asp?id=2212. Acedido em 24 Abril 2010.

SHAN, Tony; HUA, Winnie. "Solution Architecture for N-Tier Applications." *In IEEE International Conference on Services Computing, 2006. SCC '06, pp. 349-356.* IEEE CONFERENCES, 2006.
SourceForge. *kSOAP 2.* 2006. Disponível em http://ksoap2.sourceforge.net/. Acedido em 13 Fevereiro 2010.

SUI-HUI, Zhu; NI-NI, Rao. "Design and Implementation of HL7 V3 Gateway." Journal of Electronic Science and Technology of China, Vol. 3 No.2, 2005.

Sun Microsystems. *Connected Limited Device Configuration Specification Version 1.1, Java™ 2 Platform, Micro Edition*. 2003. Disponível em http://java.sun.com/products/cldc/. Acedido em 23 Abril 2010.

Sun Microsystems. *Security and Trust Services APIs for Java™ 2 Platform, Micro Edition Developer's Guide Reference Implementation 1.0*. 2004. Disponível em http://java.sun.com/javame/reference/apis.jsp. Acedido em 16 Abril 2010.

Sun Microsystems. *Sun Java System Message Queue 4.1 Technical Overview*. 2007. Disponível em http://docs.sun.com/app/docs/doc/819-7759. Acedido em 10 Maio 2010.

W3C. *Simple Object Access Protocol (SOAP) 1.1*. 2000. Disponível em http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. Acedido em 22 Novembro 2009.

W3C. *Web Services Arquitecture*. 2004. Disponível em http://www.w3.org/TR/ws-arch/. Acedido em 22 Novembro 2009.

Wikipédia. *Criptografia*. 2010. Disponível em http://pt.wikipedia.org/wiki/Criptografia. Acedido em 11 Março 2010.

Wikipédia. *Cryptographic hash function*. 2010. Disponível em http://en.wikipedia.org/wiki/Cryptographic_hash_function#cite_note-12. Acedido em 20 Março 2010.

Wikipédia. *NetBeans*. 2010. Disponível em http://pt.wikipedia.org/wiki/NetBeans. Acedido em 24 Abril 2010.

Wikipédia. *Transport Layer Security*. 2010. Disponível em http://en.wikipedia.org/wiki/Transport_Layer_Security. Acedido em 11 Março 2010.

YU, Haobo; ESTRIN, Deborah; GOVINDAN, Ramesh. "A Hierarchical Proxy Architecture for Internet-scale Event Services." *In IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1999. (WET ICE '99) Proceedings, pp. 78-83.* IEEE CONFERENCES, 2006.

Anexos

A – Reference Information Model

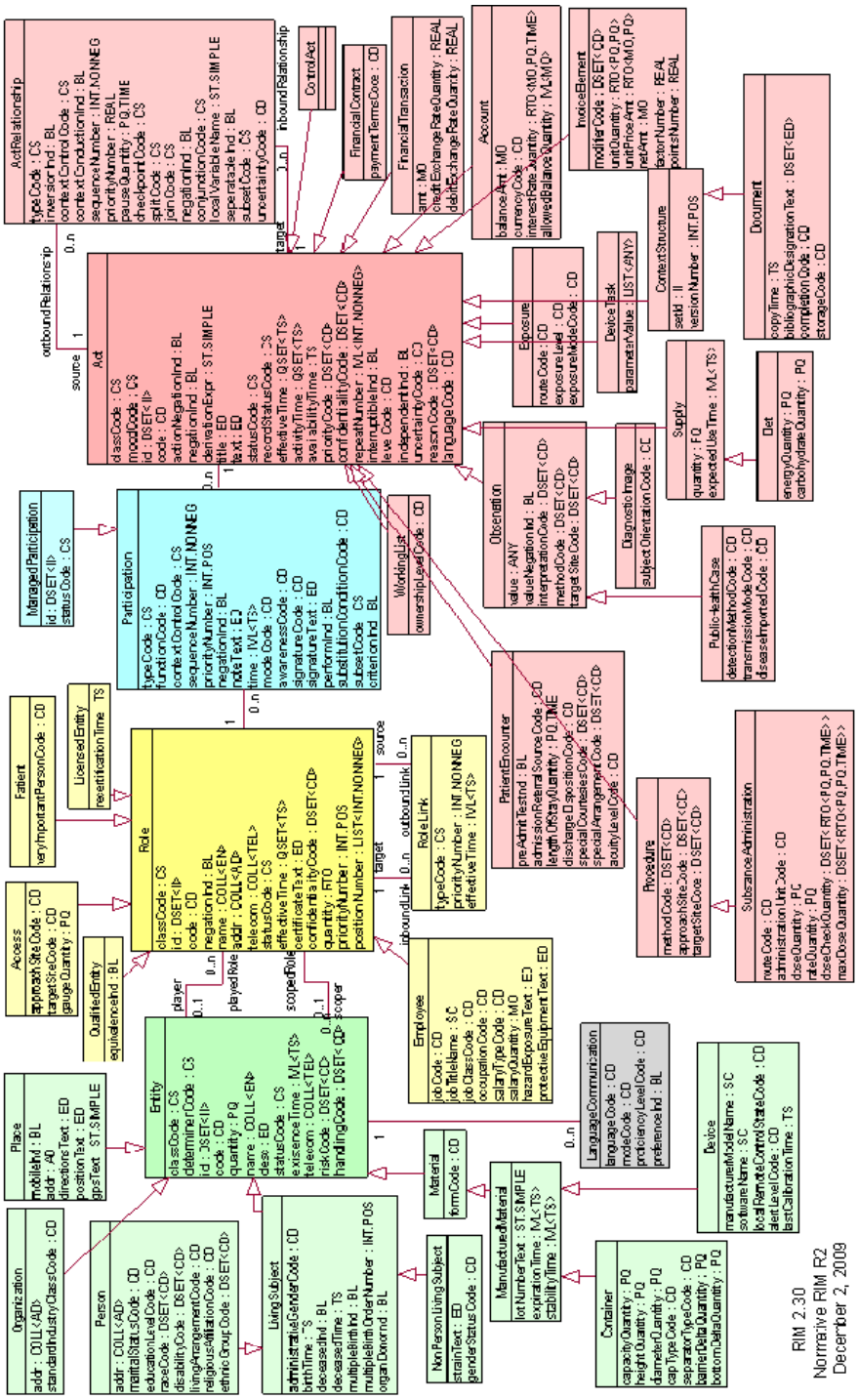

Figura 40 – Diagrama de classes RMI 2.30 (HL7, 2009)