UNIVERSIDADE FEEVALE

DAVI WINTER

PROPOSTA DE MODELO DE SISTEMA PARA CONSULTA DE DADOS MÉDICOS EM DISPOSITIVO MÓVEL COM AUXÍLIO DA TECNOLOGIA RFID

Novo Hamburgo
2014

DAVI WINTER

PROPOSTA DE MODELO DE SISTEMA PARA CONSULTA DE DADOS MÉDICOS EM DISPOSITIVO MÓVEL COM AUXÍLIO DA TECNOLOGIA RFID

Trabalho de Conclusão de Curso
apresentado como requisito parcial
à obtenção do grau de Bacharel em
Sistemas de Informação pela
Universidade Feevale

Orientador: Roberto Affonso Schilling

Novo Hamburgo
2014

AGRADECIMENTOS

Gostaria de agradecer primeiro à Deus, por ter me dado a perseverança necessária para a conclusão deste trabalho e agradecer à minha família e amigos pelo apoio e compreensão das minhas ausências durante este período.

Dedico este trabalho à minha finada mãe, que sempre me incentivou a estudar e superar os desafios.

RESUMO

Diante do crescimento cada vez maior da população, do ritmo de vida cada vez mais acelerado nos tempos atuais e também do próprio aumento da desigualdade social, existem pessoas correndo algum risco de vida, sejam de causas naturais oriundas do estresse diário e maus-hábitos, de acidentes de trânsito ou onde elas acabam se tornando vítimas da própria violência. Com isso surge também a necessidade de se agilizar o socorro aos que estão num estado de emergência, levando-se em consideração o aumento da demanda e a criticidade que o momento apresenta. Para resolver parte deste problema, este trabalho propõe um modelo de sistema para otimizar o atendimento emergencial através do auxílio da tecnologia de radiofrequência (RFID), que consiste em um implante de um *chip* RFID em cada indivíduo, o qual armazenaria a sua identificação para um sistema central que conteria os seus principais dados de saúde, como uma espécie de prontuário médico. Após o momento da leitura desta identificação pessoal com o uso de um dispositivo móvel, seriam mostrados aos agentes que estivessem prestando o socorro os dados mais importantes, representando informação fundamental para a tomada de decisões mais ágeis e corretas, maximizando as chances de preservar a saúde ou a própria vida do indivíduo.

Palavras-Chave: RFID. *Chip*. Dispositivo móvel. Sistema. Web.

ABSTRACT

Given the increasing population growth, the pace of increasingly fast-paced life in modern times and also own the increasing social inequality, there are people running any risk of life are derived from natural causes of daily stress and bad-habits, traffic accidents or where they end up becoming victims of violence itself. With this also comes the need to expedite relief to those who are in a state of emergency, taking into account the increase in demand and criticality that the moment presents. To solve part of this problem, this paper proposes a system model to optimize emergency care through the aid of radio frequency identification (RFID) technology, which consists of an implant an RFID chip in each individual, which would store your ID for a central system that would contain your main health data, as a kind of medical records. After the time of reading this personal identification with the use of a mobile device, would be shown to the agents who were providing the help the most important data, representing critical to making correct decisions more agile and information, maximizing the chances of preserving the health or one's own life.

Keywords: RFID. Chip. Mobile. System. Web.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AC	Alternating Current
AES	Advanced Encryption Standard
AMD	Advanced Micro Devices
API	Application Programming Interface
APK	Android Application Package
ARM	Advanced RISC Machine
ASP	Active Server Pages
CBC	Cipher Block Chaining
CEPT	European Conference of Postal and Telecommunications Administrations
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPF	Cadastro de Pessoa Física
DC	Direct Current
DER	Diagrama de Entidade Relacionamento
DEX	Dalvik Executable
DNS	Domain Name Service
EPC	Electronic Product Code
ER	Entidade-Relacionamento
ERP	Enterprise Resource Planning
EUA	Estados Unidos da América
FTP	File Transfer Protocol
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IFF	Identify Friend-or-Foe
ID	Identificação
IP	Internet Protocol
IPC	Inter-Process Communication
ISM	Industrial, Scientific and Medical
ISO	International Organization of Standards

JSON	JavaScript Object Notation
JSP	JavaServer Pages
LF	Low Frequency
LOOP	Looping
MD5	Message-Digest algorithm 5
MIME	Multi-Purpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
NFS	Network File System
NIST	National Institute of Standards and Technology
NSAPI	Netscape Server API
OHA	Open Handset Alliance
OPT	Option
OSI	Open Systems Interconnection
OTG	On-The-Go
PHP	Hypertext Preprocessor
PSK	Phase Shift Keying
REST	Representational State Transfer
RF	Rádio Frequência
RFCs	Request for Comments
RFID	Radio-Frequency Identication
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SO	Sistema Operacional
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SUS	Sistema Único de Saúde
TCP	Transmission Control Protocol
Transponder	Transmitter-responder
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UHF	Ultra High Frequency
UID	Unique Identification Number

UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Services Description Language
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language

SUMÁRIO

INTRODUÇÃO

O problema e uma solução proposta

Uma das grandes preocupações que existem por parte da área médica, seja nos hospitais, postos de saúde e também por parte dos socorristas é, nas situações emergenciais, não dispor das informações essenciais do indivíduo que sofreu algum tipo de dano à sua saúde. Esta situação pode se dar através de seu envolvimento em algum acidente de trânsito, acidente domiciliar, vítima de qualquer tentativa de dano corporal por parte de terceiros ou qualquer tipo de doença que represente risco à saúde desta pessoa, onde a vítima possa estar impossibilitada de fornecer informações básicas sobre ela aos agentes que prestam socorro, como o seu tipo sanguíneo, por exemplo.

Além do tipo sanguíneo, é importante e conveniente os agentes de saúde terem em mãos mais informações sobre este indivíduo, como o caso de ter algum tipo de doença que possa ser de importante relevância na hora de prestar o socorro, que se não for considerada, pode representar um risco à saúde desta pessoa, como por exemplo, sendo portadora de alguma doença cardíaca.

Diante destas circunstâncias, em conjunto com fatores como a pessoa que necessita de ajuda médica estar parcial ou totalmente impossibilitada de fornecer dados aos socorristas, também aliado ao possível fato dela poder estar desprovida de documentações referentes à sua saúde, podendo existir a possibilidade de estar sozinha ou as pessoas em seu meio não possuírem essas informações, surge a dúvida de como ela poderia ser auxiliada para minimizar seus riscos de ficar com algum tipo de sequela ou vir a se tornar uma vítima fatal pelos minutos ou segundos críticos que os agentes de saúde necessitam para dispor de informações estratégicas para salvá-la.

Neste trabalho será oferecida uma proposta que poderia contornar este tipo de problema, onde se propõe a apresentar um modelo de solução para identificar o indivíduo que será socorrido através da tecnologia *Radio-Frequency Identification* (RFID), cuja cada pessoa teria implantado sob a pele um *chip* RFID, do tamanho aproximado de um grão de arroz. Este *chip* conteria apenas um número que a identificasse (com o seu CPF, por exemplo), onde através da leitura deste *chip* RFID, seria realizada a sua identificação (criptografada), cujo dado seria disponibilizado a um dispositivo móvel (*tablet* ou *smartphone*), com acesso à internet.

Neste *chip* estaria contido a parte cliente de um sistema central que armazenaria múltiplas informações sobre a saúde de um indivíduo, como por exemplo, seu tipo sanguíneo, doenças crônicas, tipos de medicamentos, como uma espécie de prontuário médico. Após o sistema do dispositivo móvel ler e enviar a identificação da pessoa, receberia deste sistema central todos os dados do prontuário dela, contribuindo e facilitando o trabalho dos agentes de saúde no momento que está sendo prestado o socorro.

Metodologia

Quanto a pesquisa a ser realizada, do ponto de vista objetivo, será adotada em sua maior parte a pesquisa explicativa, da qual Prodanov e Freitas (2013) comentam que através de registros, análise, classificação e interpretação dos fenômenos observados, o pesquisador procura responder aos porquês das coisas.

Este trabalho irá consistir, no ponto de vista de procedimentos técnicos, em uma pesquisa bibliográfica, que possui material já publicado em livros, revistas, periódicos, artigos, trabalhos acadêmicos, entre outros, com o objetivo de deixar o pesquisador em contato direto com o assunto a ser pesquisado (Prodanov e Freitas, 2013).

Esta pesquisa bibliográfica abordará as tecnologias que são necessárias para que o modelo de sistema proposto possa ser passível de implementação e funcionamento. Entre as principais tecnologias a serem pesquisadas estará o RFID, responsável pela identificação do indivíduo, do qual será pesquisado tanto o *chip* a ser lido quanto o seu leitor.

Outras tecnologias a serem analisadas serão a comunicação do dispositivo móvel com a base de dados em um servidor central, envolvendo os tipos de serviço mais utilizados para este fim, como no uso de *Web Service* (interface utilizada para integrar e comunicar-se com sistemas diferentes), por exemplo, utilizado tanto para o envio quanto recebimento dos dados através da linguagem padrão *Extensible Markup Language* (XML).

Na questão do sistema do dispositivo móvel, será necessário explorar como funciona à nível de aplicação e comunicação a plataforma a ser usada, neste caso o *Android*, que terá o sistema com a interface para o usuário final (o agente de saúde) incorporado a ele.

1. ESTUDO SOBRE A TECNOLOGIA RFID

Neste capítulo será abordado o estudo sobre a tecnologia RFID, apresentando a sua história, características e modo de funcionamento.

1.1 História

A tecnologia RFID não é algo recente, passou a ser empregada pouco antes da II Guerra Mundial pelas forças armadas norte-americanas, em virtude da dificuldade de localizar alvos terrestres, aéreos e marítimos, onde criaram o método de identificação IFF, que possibilitava a distinção entre as aeronaves aliadas e as inimigas (SHAHRAM, 2005 apud OLIVEIRA; PEREIRA, 2010). Com isso, este sistema de identificação passou a ser a base dos sistemas de controle do tráfego aéreo (OLIVEIRA; PEREIRA, 2010).

Com o avanço da tecnologia, surgiram melhoras no alcance e na frequência de operação dos radares, a qual consistia no uso de ondas mais curtas para possibilitar um maior alcance, denominado como radar de micro-ondas. Esta evolução tecnológica proporcionou mais recursos aos militares, como a possibilidade de seguir as rotas de navios e aviões, calcular a sua velocidade, além de identificá-los como objeto aliado ou inimigo (CASTRO, 2012).

Entre as décadas de 50 e 60 foi consolidado um período de explorações da RFID e realização de experimentos laboratoriais (GOMES, 2007 apud CASTRO, 2012), que ficou caracterizado pela exploração de técnicas para o seu uso, acompanhado de sucessivos avanços tecnológicos para rádio e radar, sendo um período em que estavam em pauta diversas tecnologias relacionadas a esta tecnologia (PASSARETTI, 2010).

Já entre os anos 60 e 70, com mais avanços em relação à RFID, começaram a ser desenvolvidas novas aplicações para a tecnologia, como a implementação de sistemas de controle de produtos, a qual consistia no uso de etiquetas de 1 bit para a identificação de itens que saíam da loja, para ser verificado se determinado item havia sido pago ou roubado. Se o item fosse pago, este bit seria zerado junto ao caixa e na saída o alarme não dispararia. Caso fosse roubado, a detecção da etiqueta com o bit guardado faria o alarme soar. Neste período também surgiram etiquetas em formato de cartão, para fins de controle de acesso.

O surgimento da primeira etiqueta RFID se deu em 1969, quando a IBM possuía um sistema óptico chamado CARTRAK para os comboios. A ideia inicial seria este

sistema realizar a leitura de um código de barras na lateral de cada caminhão, onde passariam a ser identificados. Mas este sistema se tornaria ineficiente devido a fatores como sujeira, luz intensa ou danos na própria etiqueta do código de barras. Diante destas deficiências, a IBM projetou um sistema que possuiria um transmissor, um receptor, uma pequena quantidade de memória interna e uma fonte de alimentação, dando origem mais tarde a um sistema de pagamento automático de portagens (CASTRO, 2012).

Na década de 70, o governo dos EUA passa a mostrar maior interesse pela tecnologia RFID, onde o laboratório nacional de Los Alamos recebeu um pedido do departamento de energia para realizar o rastreamento de materiais nucleares. O plano consistia em colocar um *transponder* em cada caminhão transportador, que continha duas informações, sendo a primeira do caminhão e a outra do motorista (BASTOS; SILVA, 2010).

No ano de 1973, foi requerida por Mario W. Cardullo uma patente para uma etiqueta ativa de RFID com uma memória regravável. Charles Walton, um empreendedor da Califórnia, recebeu a patente por um *transponder* passivo para destravar uma porta sem o uso de chave, que consistia em um cartão com este *transponder* que comunicava-se com um leitor na porta, que era destravada ao identificar o número do cartão como válido (CARDULLO, 2002 apud BASTOS; SILVA, 2010).

Ainda nesta década, surge o aparecimento das primeiras RFID para animais, do qual:

> "Rossing (1976, 1978) relatou experimentos nos anos 70 com *transponders* eletrônicos utilizados na alimentação individual de vacas e no registro automático de dados. A primeira geração dos *transponders*, as 'caixas pretas' eletrônicas eram anexadas em colares colocados no pescoço dos animais." (DE CF MACHADO; NANTES, 2004).

Nos anos 1980, o MIT, junto com outros centros de pesquisa, começou a estudar uma arquitetura que utilizasse recursos das tecnologias baseadas em radiofrequência para servir como um modelo para o desenvolvimento de novas aplicações de rastreamento e localização de produtos. Com estes estudos, criou-se o EPC, oferecendo uma arquitetura que possibilitasse a identificação de produtos que utilizassem os recursos existentes nos sinais de radiofrequência (BERNARDO, 2004).

Ainda na década de 80, passou-se a investir em aplicações comerciais para sistemas de gestão pecuária, sistema de entrada sem uso de chaves, além de acesso

pessoal. Na época, os sistemas implementados eram proprietários, não existindo uma padronização, que, aliada à pouca concorrência entre as indústrias do ramo, tornava os custos elevados por cada etiqueta.

Nos anos 90, a IBM criou e patenteou uma tecnologia de RFID no formato UHF, que possibilitava uma alcance de leitura de aproximadamente 6 metros e uma transferência de dados mais rápida, passando a ser utilizadas em barreiras eletrônicas nas estradas.

Algumas empresas dos EUA e Europa passaram a investir na tecnologia RFID, como a Philips, Mikron, Alcatel e Bosch e empresas como a IBM, AMD, Intel e Motorola possibilitaram a produção mais barata destas etiquetas, tornando-as viáveis economicamente. Organizações como a CEPT, a ISO e o Auto-ID Center no MIT uniram esforços para a criação de uma padronização da tecnologia RFID.

Nos anos 2000, estudou-se a viabilidade do preço das etiquetas à US$ 0,05, possibilitando a substituição do código de barras. Em 2003, a Wal-Mart, determinou a utilização em massa da RFID até 2005, e em 2007, passa a se aproximar dos US$ 0,05 na compra de grandes volumes (LANDT et al, 2001 apud RODRIGUES, 2010).

1.2 Funcionamento do sistema RFID

Um sistema RFID consiste no uso do espectro eletromagnético para realizar a transmissão da informação, da qual utiliza radiofrequência para efetuar a troca de dados, possibilitando o armazenamento ou a recuperação de dados de forma remota em um dispositivo denominado *transponder* (ou etiqueta), que pode estar inserido em algum produto, bem ou até em um ser vivo (CASTRO, 2012).

Quando um leitor interroga uma etiqueta, lhe é devolvida a informação, podendo variar desde um bit até um conjunto de dados que ficam armazenados no *microchip*. Quando são etiquetas ativas, não necessitam da presença de um leitor para a transmissão da informação (MEIRELES, 2009).

O leitor atua como o principal componente no sistema de RFID, pois realiza a comunicação entre os sistemas externos com as etiquetas, o qual fica encarregado pela gestão do sistema, de efetuar controles de múltiplos acessos das etiquetas, rejeição de repetições de dados, correção de erros, entre outros. Todos estes mecanismos para o

controle do sistema, segurança e gestão são implementados no leitor devido às etiquetas se tratarem de dispositivos simplificados e de baixo custo (GOMES, 2007).

Quando um leitor emite um sinal questionando as etiquetas no seu raio de alcance, as etiquetas aproveitam a energia deste sinal para recarregarem seus circuitos, emitindo de volta um sinal contendo a informação guardada no seu *microchip*, cujo leitor capta o sinal emitido pela etiqueta, encaminhando a informação para o restante do sistema, como para um computador, por exemplo.

A comunicação através de RFID se dá de modo assimétrico, do qual o leitor tem a tarefa de transmissor e a etiqueta o de retransmissor, o que explica o sucesso da RFID, fazendo com que a etiqueta dispense a necessidade de criar o seu próprio sinal, apenas realizando a retransmissão do sinal emitido pelo leitor, possibilitando o uso de etiquetas bastante simples, de tamanho reduzido, as quais propiciam a sua produção a custos baixos (MEIRELES, 2009).

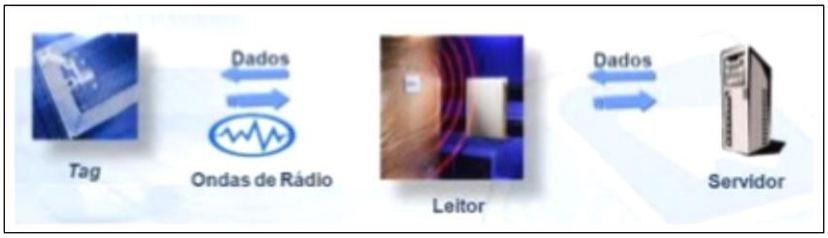


Figura 1.1 Etiqueta, leitor e servidor
Fonte: Luís (2009) apud Castro (2012)

1.2.1 Faixas de Frequências

A característica inerente aos sistemas RFID é o fato destes sistemas emitirem ondas eletromagnéticas, na qual são englobados como sistemas de radiofrequência (RF). Devido a isso, precisa-se que sejam estipuladas as faixas de frequência para que outras frequências de rádio não causem interferências no sistema de RFID, assim como o sistema de RFID não pode causar interferências nos sistemas de celular, rádio e televisão (OLIVEIRA; PEREIRA, 2010).

Para o funcionamento do sistema RFID, as bandas de operação estão relacionadas as bandas ISM, pois apesar destas bandas não exigirem licença de uso, os intervalos do espectro são muito concorridos, necessitando pra isso um controle apertado por parte das entidades reguladoras de cada país. Como este controle ocorre a nível nacional, surgem dificuldades para chegar a um consenso para o uso das bandas de

frequência de uma determinada tecnologia. Abaixo, no Quadro 1.1 encontram-se alguns aspectos das diversas frequências utilizadas pelos sistemas de RFID:

Quadro 1.1: Bandas de frequência RFID

Frequência	Regulação	Alcance típico	Vantagens	Comentários
< 135 kHz	Banda ISM, alta potência	<10cm (passivo)	Boa penetração em líquidos	Access Control Animal tagging
6.78 MHz 13.56 MHz 27.125 MHz	Banda ISM, regulação praticamente igual em todo o mundo	<1m (passivo)	Penetração média em líquidos	Smart Cards, Access Control, Imobilização de veículos
433 MHz	Banda ISM para dispositivos de comunicação de curto alcance, Banda não uniforme	<100m (activo)	Funciona bem em ambientes com metais	Tags activas
888-956 MHz	Banda não uniforme mundialmente	<10m (passivo US) <4m (passivo UE)	O melhor alcance para comunicações passivas	Normas Wal-Mart, DoD
2.45 GHz	Banda ISM	<3m (passivo) <50m (SAW)	Alternativa Para os 900 MHz	Wi-Fi, Bluetooth
5.4-6.8 GHz 24,05 - 24,5GHz	Bandas salvaguardadas para uso futuro			

Fonte: Gomes (2007) apud Meireles (2009)

Mesmo as variações nacionais e das variadas bandas utilizadas nos sistemas de RFID, pode-se agrupá-los em três grupos de frequência de operação, observando que há aspectos similares. Portanto, os sistemas de baixa frequência (LF) operam na faixa de 100KHz à 500KHz, os sistemas de alta frequência (HF), trabalham entre os 6.78MHz e 433MHz, já os sistemas de ultra alta frequência (UHF) ultrapassam os 888MHz de frequência.

No que diz respeito ao alcance, trata-se de outro fator a ser considerado nos sistemas de RFID. Sendo assim, encontra-se sistemas de baixo alcance, normalmente até 10cm e que operam nas frequências inferiores à 135KHz (MEIRELES, 2009).

Este alcance é muito limitado principalmente devido ao pequeno ganho das antenas, pois como os comprimentos de onda são muito grandes nas baixas frequências, estas são maiores que as dimensões das antenas das etiquetas, pois o comprimento da antena tem proporção direta com o seu ganho de sinal, tendo relação com o comprimento da onda, resultando em um ganho muito pequeno em baixas frequências.

Geralmente, esta faixa de frequência emprega o uso de etiquetas passivas, na qual possui uma baixa taxa de transferência de dados (na ordem dos Kbits/s) entre o leitor e a etiqueta, onde são recomendadas para ambientes que contenham metais, líquidos, sujeira, neve ou barro, compreendendo a maioria das aplicações mundiais, pois a faixa LF é aceita no mundo todo (PASSARETTI, 2010).

Já os sistemas de médio alcance, trabalham em torno dos 15MHz (MEIRELES, 2009), cuja frequência típica é 13,56MHz. Os sistemas que operam nesta frequência geralmente utilizam etiquetas passivas, possuem uma baixa transferência entre o leitor e a etiqueta, apresentando um desempenho satisfatório quando inserido em ambientes com materiais líquidos e metálicos.

Sistemas com esta frequência são largamente usados, especialmente por esta frequência ter regulamentação no mundo todo, podendo ser encontradas aplicações em muitas áreas, com destaque na implantação em hospitais, pois sua frequência típica não causa interferência no funcionamento de equipamentos médicos (PASSARETTI, 2010).

Por fim, existem os sistemas de alto alcance que utilizam as frequências de micro-ondas e UHF (MEIRELES, 2009), na qual as micro-ondas podem alcançar taxas de transferência na casa dos Mbits/s. Contudo, pelo fato de prováveis danos que as altas frequências podem causar a saúde humana, foram impostos limites de potência pelos órgão reguladores aos sistemas que operam com micro-ondas e UHF, reduzindo a distância de leitura por volta de 3 à 9m no uso de etiquetas passivas (HUNT, 2007 apud PASSARETTI, 2010).

A arquitetura e o tipo de etiqueta influenciam muito no alcance da frequência em uso, cujo alcance das etiquetas ativas é muito maior que o alcance das passivas, pois num cenário de etiquetas passivas, surgem duas restrições básicas relacionada ao alcance, sendo uma delas a dependência da existência de um sinal potente para carregar a etiqueta e o outro quanto a necessidade de que haja uma pequena potência na etiqueta para que possa responder, minimizando a distância entre a etiqueta e o leitor. Já num sistema de RFID ativo, a limitação da potência do sinal recebido não possui tanta dependência, permitindo a comunicação em até 100m (MEIRELES, 2009).

1.2.2 Padronização

O objetivo da padronização é determinar os aspectos de funcionamento e operação de equipamentos para que possam ser produzidos dispositivos com aspectos comuns através de fabricantes distintos. Portanto, é através da padronização que diversas tecnologias convivem em um mesmo ambiente.

Quanto aos fabricantes de RFID, os maiores neste segmento disponibilizam sistemas proprietários, resultando numa grande variedade de protocolos e de sistemas. Porém, muitas entidades inseridas em projetos de tecnologias RFID estão trabalhando em parceria com muitos estudiosos e fabricantes da tecnologia na busca da padronização de protocolos e regras para seu uso.

Dentre as principais entidades para a determinação de padrões da RFID, encontra-se a ISO, na qual esta entidade envolve 148 países (FINKECZELLER, 2003 apud OLIVEIRA; PEREIRA, 2010). A seguir, no Quadro 1.2 está listado os padrões para a tecnologia RFID, publicados pela ISO:

Quadro 1.2: Padrões ISO para a tecnologia RFID

ISO Standard	Título	Status
ISO 11784	RFID para animais – estrutura de código	Publicado em 1996
ISO 11785	RFID para animais – concepção técnica	Publicado em 1996
ISO/IEC 14443	Identificação de cartões – cartões com circuitos integrados sem contato – cartões de proximidade	Publicado em 2000
ISO/IEC 15693	Identificação de cartões – cartões com circuitos integrados sem contato –	Publicado em 2000

	cartões de vizinhança	
ISO/IEC 18001	Tecnologia da Informação – Gerenciamento de Itens de RFID – Perfil de Requisitos de Aplicação	Publicado em 2004
ISO/IEC 18000-1	Parâmetros Gerais para Comunicação por Interface por Ar para Freqüências Globalmente Aceitas	Publicado em 2004
ISO/IEC 18000-2	Parâmetros para Comunicação por Interface por Ar abaixo de 135 kHz	Publicado em 2004
ISO/IEC 18000-3	Parâmetros para Comunicação por Interface por Ar em 13,56 MHz	Publicado em 2004
ISO/IEC 18000-4	Parâmetros para Comunicação por Interface por Ar em 2,45 GHz	Em Revisão Final
ISO/IEC 18000-6	Parâmetros para Comunicação por Interface por Ar em 860 a 930 MHz	Publicado em 2004
ISO/IEC 15961	Gerenciamento de Itens de RFID – Protocolo de Dados: Interface de Aplicação	Publicado em 2004
ISO/IEC 15962	Gerenciamento de Itens de RFID – Protocolo: Regras de Codificação de Dados e Funções de Memória Lógica	Publicado em 2004
ISO/IEC 15963	Gerenciamento de Itens de RFID – Identificação única do RF Tag	Em Revisão Final

Fonte ISO (2006) apud Oliveira; Pereira (2010).

Os padrões definidos pela ISO para a tecnologia RFID engloba as áreas que definem os padrões de identificação com relação à codificação do *ID Number* e demais informações inseridas nas etiquetas, os protocolos de interface aérea que estabelecem as regras para a comunicação entre etiquetas e leitores, os protocolos utilizados no *middleware*, padrões para testes, tendências e segurança (SANGHERA, 2007 apud PASSARETTI, 2010).

Quanto aos protocolos de interface aérea, são determinadas as regras para a comunicação entre etiquetas e leitores que envolvem a codificação dos dados, modulação e demodulação, comandos a serem executados na etiqueta como leitura, escrita, modulação dos dados, bloqueio de informações, destruição da etiqueta, além de contemplar algoritmos anti-colisão.

A ISO desenvolve também padrões para aplicações da RFID, sendo um deles para o rastreamento animal, com uso de baixa frequência. A ISO possui dois padrões para essa finalidade, sendo o ISO 11784, que determina a estrutura do código para etiquetas usadas em animais, cuja identificação destes ocorre pelo código do país e um único ID

nacional e o ISO 11785, que estabelece os parâmetros técnicos para a comunicação entre leitor e etiqueta (PASSARETTI, 2010). Entre as normas ISO para a RFID, estas duas possivelmente são as que mais se aproximam do modelo proposto neste trabalho.

Contudo, EPCGlobal é o órgão padronizador específico da tecnologia RFID, sendo este órgão uma *join-venture* entre a GS1 e a GS1 US, no qual conseguiu criar o padrão EPCGlobal Gen 2, popularmente conhecida por Gen 2, em dezembro de 2004, revolucionando a tecnologia RFID no avanço pela padronização das etiquetas, que atualmente é um dos principais problemas na tecnologia (SANGHERA, 2007 apud PASSARETTI, 2010).

1.3 Estrutura do RFID

A arquitetura de um sistema RFID é composta em sua essência por um *transponder* (ou etiqueta), um leitor que fará a leitura desta etiqueta e um *middleware* (hardware e software) para receber os dados lidos.

1.3.1 Etiquetas RFID

Existem diversos tipos de etiquetas, podendo elas serem do tipo passiva, semi-passiva ou ativa.

- **Etiquetas Passivas**

As etiquetas passivas se caracterizam por não possuírem qualquer tipo de alimentação (RAMOS, 2007; SILVA, 2006 apud CASTRO, 2012). São ativadas pela potência emitida pelo leitor, realizando o uso desta mesma energia para transmitir a sua informação (SYBASE, 2006 apud CASTRO, 2012). Em função de não possuírem bateria, podem assumir tamanhos cada vez menores e são facilmente moldáveis, além de não precisarem receber manutenção por um longo período de tempo.

São constituídas basicamente por silício, existindo atualmente etiquetas feitas por materiais poliméricos. Devido a estas facilidades, pretende-se no futuro que este tipo de etiqueta possa ser impressa, assim como um código de barras, tornando-se economicamente comparável ao código de barras (COUTO apud CASTRO, 2012).

> "Por não usar uma bateria a vida destas etiquetas é quase ilimitada e não necessita de muita manutenção. Consegue suportar condições adversas sem se danificar, são pequenas e finas e têm custos de produção baixos." (MEIRELES, 2009).

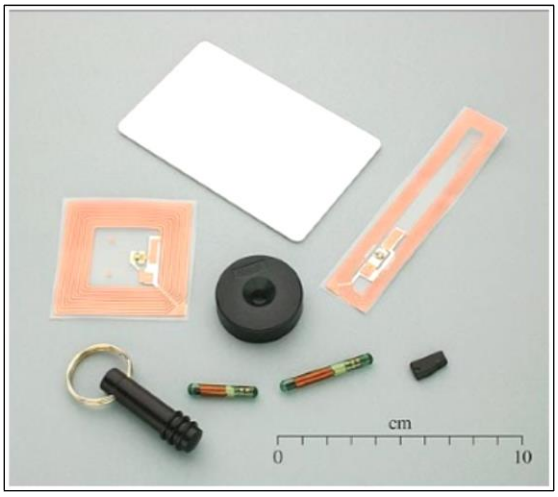


Figura 1.2: Exemplos de etiquetas passivas
Fonte: Meirelles (2009)

- **Etiquetas Ativas**

Diferentemente das etiquetas passivas, as etiquetas ativas possuem fonte de alimentação própria capaz de alimentar os seus circuitos para poder fazer a recepção e/ou a transmissão dos dados com o leitor, não sendo necessária a energia de um leitor. Devido a isso, sua estrutura é mais complexa, com a vantagem que seu alcance é muito maior que o de uma etiqueta passiva.

Possuem a vantagem de ter grande capacidade de armazenamento, oferecem maior processamento de dados, segurança mais sofisticada e contam com suporte a componentes externos como sensores, possibilitando a execução de funções extras.

Apesar de poderem suportar sensores e realizar tais funções extras, podendo atuar como um transmissor para o leitor, esta não é uma prática aconselhada, pois consome rapidamente com a bateria. Apresentam a desvantagem de serem mais caras, possuem maior fragilidade sob condições adversas e exigem manutenção recorrente, como para recarregar a bateria, por exemplo (KAYA; KOSER; GOMES, 2007 apud MEIRELES, 2009).

Figura 1.3: Exemplos de etiquetas ativas
Fonte: Couto, Dias (2008) apud Castro (2012)

- **Etiquetas Semi-Passivas**

Este tipo de etiqueta apresenta características tanto das etiquetas passivas quanto das etiquetas ativas. Com capacidade de ter alimentação interna, esta serve apenas para alimentar os circuitos internos e não para emitir um novo sinal de RF para um leitor. Assemelha-se à etiqueta passiva no que diz respeito à sua antena e quanto ao modo de funcionamento, pois sempre necessita de um leitor para se comunicar. Por ter alimentação interna, pode possuir um *microchip* com capacidade superior (GOMES, 2007).

1.3.2 Leitores RFID

Os leitores são os responsáveis por fazer o "meio-campo" entre a etiqueta e o software, exercendo funções como a leitura das etiquetas, escrita nas etiquetas, além de alimentar as etiquetas passivas. O leitor constitui-se como o componente central do hardware de um sistema RFID, oferecendo conexão e controle de suas operações para os dispositivos que se integram a ele. Por apresentarem maior complexidade na sua constituição e a variedade de componentes internos, os leitores oferecem menor resistência às condições adversas do ambiente em relação as etiquetas (PASSARETTI, 2010).

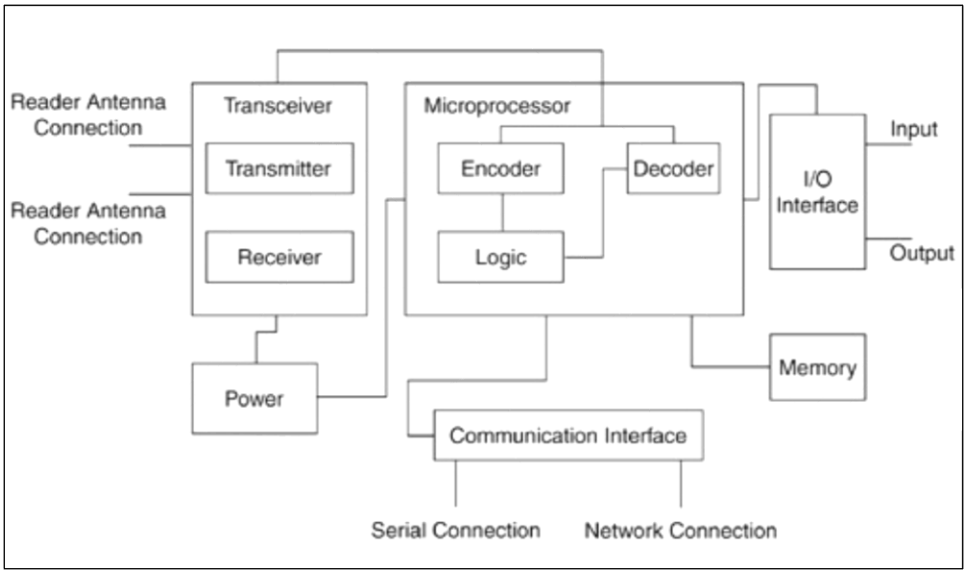


Figura 1.4: Componentes de um leitor de RFID
Fonte: Lahiri (2006) apud Passaretti (2010)

Os principais componentes de um leitor são:

- **Transmissor:** É o componente responsável por transmitir energia AC e o ciclo de *clock*, através das antenas do leitor para as etiquetas, na zona de leitura, além de, principalmente, emitir o sinal do leitor para o ambiente e captar a resposta das etiquetas, recebidas pelas antenas do leitor (LAHIRI, 2006 apud PASSARETTI, 2010).

- **Receptor:** Tem a função de receber os sinais analógicos enviados pelas etiquetas, encaminhando estes sinais ao microprocessador, no qual serão convertidos em sinais digitais (PASSARETTI, 2010).

- **Microprocessador:** Possui a finalidade de implementar o protocolo de leitura para realizar a comunicação com as etiquetas que são compatíveis, além de efetuar a decodificação e realizar a análise de erros dos sinais analógicos recebidos pelo módulo receptor.

- **Memória:** Responsável por armazenar os dados, parâmetros de configuração do leitor e as leituras das etiquetas que foram realizadas, na qual assegura todas (ou parte) das leituras numa situação onde ocorra a queda de conexão entre o leitor e o controlador, ou do próprio sistema (LAHIRI, 2006 apud PASSARETTI, 2010).

- **Canais de entrada / saída de sensores externos:** Possibilita a conexão de sensores a estes leitores, pois um sensor pode, através de algum evento externo, ligar ou

desligar um leitor, contribuindo para a economia de energia, uma vez que o leitor será ativado somente quando solicitado (HUNT, 2007 apud PASSARETTI, 2010).

- **Controlador:** Tem a função de permitir a comunicação do leitor com uma entidade externa, como um software, por exemplo, para controlar funções do leitor. É somente através deste componente que é possível realizar alguma comunicação com o leitor dentro de um sistema RFID. Este controlador pode estar embutido dentro do próprio leitor (firmware) ou pode estar separado.

- **Interface de comunicação:** Por meio do controlador, este componente utiliza as instruções de comunicação com o leitor, permitindo interação com entidades externas, seja através de transferência de dados armazenados ou para receber comandos e enviar as respectivas respostas. Deve possuir interfaces de rede como serial, Ethernet ou Wireless.

- **Fonte de alimentação:** Responsável pelo fornecimento de energia para o leitor.

- **Antena:** É o componente que transmite os dados entre a etiqueta e o leitor. O formato da antena, assim como a sua localização, são determinantes para a definição de parâmetros como a cobertura, alcance e eficiência da comunicação. Estes parâmetros devem ser levados em conta de acordo com o tipo de aplicação para o qual o leitor será utilizado (PASSARETTI, 2010).

Figura 1.5: Exemplo de leitor de RFID
Fonte: Passaretti (2010)

1.3.3 *Middleware*

O *middleware* é parte fundamental em um sistema RFID. Tem a função de atuar como uma interface entre o sistema RFID e a aplicação com a qual os dados se integrarão. Como as etiquetas RFID contribuem significativamente para o aumento na quantidade e

tipos de dados disponíveis para uma organização, é necessário que exista uma capacidade de categorizar, processar e também equilibrar a carga de dados. Numa situação onde, por exemplo, uma etiqueta passa a ser lida duas vezes, um software de *middleware* pode agir como um filtro limpando e sincronizando os dados da etiqueta, evitando que esta duplicação seja repassada para as aplicações.

Como os dados da etiqueta podem, às vezes, serem lidos de forma incorreta, através do uso de uma lógica embutida, o software do *middleware* pode corrigir, agregar ou filtrar os dados da etiqueta para contornar este problema. Através de um *middleware* é possível coordenar as atividades de vários leitores, fazendo com que itens ou caixas com etiquetas passem a partir do alcance de leitura de um leitor para o outro, além do mais, o software do *middleware* pode servir de ferramenta que permitirá a captura e análise da atividade da etiqueta dentro de uma organização.

Os dados de RFID só são úteis se puderem ser reconhecidos e processados. Ocorre que hoje a maioria dos aplicativos *back-end* não está habilitada para a integração de dados com a RFID, portanto, é necessário modificar o software de ERP de uma organização para acomodar os dados adicionais disponibilizados via tecnologia RFID (JILOVEC, 2004).

A importância de um *middleware* se dá pela sua capacidade de absorver diferenças entre diversos leitores, possibilitando a integração dos dados, oferendo flexibilidade e escalabilidade ao sistema. Quando desenvolvido corretamente, acaba trazendo benefícios para uma aplicação, pois levará em consideração as necessidades de uma aplicação no que se refere à extensibilidade, portabilidade, escalabilidade, tolerância a falhas, entre outros.

Além de *middlewares* desenvolvidos de forma personalizada, existem também os que são comerciais, que oferecem capacidades para o gerenciamento de eventos, apresentando-se com capacidades e sofisticações para o controle de sensores e para lidar com o fluxo de trabalho (DIAS, 2014).

1.4 Considerações sobre a tecnologia

Neste capítulo se pôde compreender como surgiu a tecnologia RFID, inicialmente como uma aliada nos períodos de guerra, passando-se a descobrir o seu

potencial uso nas áreas industriais, comerciais, pecuária, saúde, entre outros, tornando os processos de controle das entidades que o adotaram mais rápidos e fáceis de se controlar.

Pôde-se compreender o funcionamento desta tecnologia, da qual foi explanado os processos necessários para que a captura da informação contida em uma etiqueta possa ser extraída dela até o leitor que a interrogou, passando pelas etapas de emissão, modularização e digitalização do sinal trocado. Foi possível verificar as funcionalidades que o leitor tem no processo de comunicação, contornando problemas de falhas no que diz respeito aos erros ou redundância de informação.

Foi visto que para o funcionamento com base na RFID, é necessário que se compreenda os tipos de frequência existentes, sendo cada tipo apropriado para determinados cenários, influenciando parâmetros como a energia, distancia, velocidade, ambientes e os tipos de componentes envolvidos no sistema. Teve-se também uma prévia sobre os protocolos e normas utilizados para regularizar e padronizar o emprego da tecnologia.

Quanto aos componentes utilizados para a dinâmica de um sistema RFID, constatou-se as categorias de etiquetas disponíveis, sendo apresentadas as suas características no que tange os seus aspectos físicos, funcionalidades, fragilidades, durabilidade, entre outros. Também foi possível entender o funcionamento básico de um leitor, sendo abordado os seus mecanismos internos em conjunto com as suas funções para efetuar o tratamento do sinal, desde sua emissão até sua recepção, além das funções adicionais.

Observou-se também, não menos importante, o *middleware*, responsável pela interface do leitor com uma entidade externa, a fim que se possa auxiliar no processo de leitura, no sentido de garantir a consistência dos dados lidos e para que se possa manipular estes dados para encaminhá-los aos sistemas externos ao sistema RFID para o proveito das informações.

2. EXPLORAÇÃO DO FUNCIONAMENTO DO SERVIDOR

Neste capítulo será abordado as principais tecnologias que envolvem o funcionamento do servidor.

2.1 Protocolos TCP/IP

Os protocolos TCP/IP tiveram a sua criação para que se pudesse viabilizar a interligação de computadores na rede mundial de computadores, ou seja, a Internet, que, devido ao seu sucesso, tanto na interligação de redes locais e de longa distância, fez com que estes protocolos passassem a se tornar um padrão de uso e utilizados em larga escala.

Estes protocolos se apresentam como uma opção altamente flexível, funcionando em diversos tipos de arquiteturas de redes, desde cabos coaxiais até redes sem fio, transmitidos por diversos canais, como fibras ópticas e linhas telefônicas, por exemplo. Com o uso destes protocolos, através da API de *sockets* (STEVENS et al., 2003 apud TEIXEIRA, 2004), se permite que as aplicações possam ser escritas com um alto grau de abstração no que se refere a como a comunicação ocorre, oferecendo um trabalho facilitado aos programadores e um aumento na portabilidade das aplicações.

Os protocolos TCP/IP oferecem uma série de vantagens aos demais protocolos, podendo-se citar de início o fato de seu uso ser um padrão aberto, no qual encontra-se disponível gratuitamente e não encontra-se atrelado a nenhuma plataforma de hardware ou software em especial, sendo possível obter as suas especificações na internet, que está disponibilizada de forma livre, através de RFCs, promulgadas pela IETF. (COMER, 2000; HUNT, 1997 apud TEIXERA, 2004).

Conforme Teixeira (2004) descreve, outro ponto a seu favor é a sua independência de rede, pois se trata de um protocolo que usa a comutação de pacotes para se comunicar, tendo o seu funcionamento suportado sobre diversos protocolos nas camadas de enlace e física, apresentando-se de forma transparente para as aplicações, tornando-se com isso a opção mais utilizada para a interligação de hardware e software de diversos tipos.

Outras das características mais vantajosas deste protocolo são no que se refere às confirmações fim-a-fim, através do mecanismo de *acknowledgements*, tanto na origem, quanto no destino final da comunicação, tornando a entrega de dados mais segura de uma maneira geral (COMER, 2000; HUNT, 1997 apud TEIXERA, 2004).

No que se refere a arquitetura TCP/IP, esta encontra-se dividida em 4 camadas, sendo elas Aplicação, Transporte, Internet e Acesso à rede. A Camada de Acesso à Rede é caracterizada por encontrar-se na camada mais baixa da hierarquia, servindo de base para a entrega dos dados entre computadores conectados diretamente. Para isto, se utiliza de diversos protocolos de acesso às camadas inferiores, escolhendo-o de acordo com o tipo de rede disponível.

É nesta camada que se pode verificar a flexibilidade que este protocolo oferece, permitindo a sua operação em diversos meios físicos. Além disso, esta camada trata das funções de encapsulamento dos datagramas IP nos frames usados na rede, além da conversão dos endereços IP para o formato da rede.

Já a Camada de Internet tem a função de efetivar a comunicação entre as máquinas da rede, sendo que nesta camada está estabelecido o protocolo IP, que fornece o serviço de entrega de pacotes na qual as redes TCP/IP foram construídas (HUNT, 1997 apud TEIXEIRA, 2004). Toda a informação numa rede TCP/IP está dentro de um datagrama IP, que é utilizado pelas camadas superiores. O protocolo IP define o formato do datagrama, que é a unidade básica de uma transmissão de dados em uma rede TCP/IP, sendo responsável pela função de endereçamento da rede, roteamento dos datagramas entre os hosts e pela fragmentação e remontagem destes datagramas quando mudam de camada dentro da pilha TCP/IP.

O protocolo IP se caracteriza por ser não orientado à conexão, ou seja, estes datagramas carregam a sua informação, permitindo a independência dos demais datagramas no processo de roteamento e não precisam estabelecer uma troca de informações de controle para iniciar a comunicação. Ainda, este protocolo IP é caracterizado como não-confiável, pois não possui mecanismos de detecção e controle de erros, ficando esta parte à cargo das camadas superiores.

No que diz respeito à Camada de Transporte, esta é responsável pela comunicação fim-a-fim entre dois *hosts*, definindo essencialmente os protocolos TCP e UDP. No protocolo TCP é fornecido o serviço de transporte orientado à conexão sobre o protocolo IP, ou seja, necessita-se o estabelecimento da conexão fim-a-fim entre os dois hosts, que se realiza através de troca de mensagens de controle (COMER, 2000 apud TEIXEIRA, 2004), no qual o TCP oferece um serviço confiável, garantindo a chegada dos dados ao seu destino livre de erros e na ordem de envio.

Além disso, realiza o controle de fluxo, evitando que um destino fique sobrecarregado devido a uma fonte muito rápida. Para o TCP, os dados são transmitidos em um fluxo de bytes, sem qualquer delimitador entre eles, fazendo que com isso seja simplificada a implementação em determinados tipos de aplicações cliente-servidor. Devido à sua confiabilidade, este protocolo é altamente difundido na internet, principalmente por ser um ambiente favorável para a ocorrência de falhas. Alguns protocolos da camada de aplicações trabalham em cima do protocolo TCP, sendo eles o FTP, TELNET e HTTP, por exemplo.

O outro principal protocolo da camada de transporte é o UDP, no qual, diferentemente do TCP, oferece um serviço de transporte não orientado à conexão sobre o protocolo IP, sendo a transmissão de dados iniciada assim que desejado, evitando a sobrecarga inicial neste processo como acontece com o TCP em virtude do estabelecimento de conexão requerida por este.

Devido a isso, o UDP não possui qualquer tipo de mecanismo para o controle de erros, sendo caracterizado como um serviço de transporte não confiável, pois esta parte ou acaba dispensada, em determinados casos, ou passa a ser implementada no nível de aplicação.

O UDP é um protocolo visto como um datagrama, denominado pacote, sendo indicado para a implementação de aplicações cliente-servidor, em especial redes locais (COULOURIS et al., 2000 apud TEIXEIRA, 2004), podendo-se encontrar estes protocolos sendo utilizados em mensagens DNS e SNMP, além de ter seu uso largamente empregado nas aplicações de transmissão de áudio e vídeo.

No que se refere à Camada de Aplicação, ela se caracteriza por possuir todos os processos que se utilizam dos serviços da Camada de Transporte para a entrega dos dados, utilizando tanto os serviços não orientados à conexão, baseado em mensagens individuais (UDP) ou por serviços orientados à conexão, que oferecem a abstração de um fluxo contínuo de bytes (TCP). Nesta camada se concentram as aplicações utilizadas pelos usuários e os tipos de protocolos utilizados nela, entre eles o TELNEL, FTP, DNS, HTTP, NFS, SMTP, entre outros (TEIXEIRA, 2004).

2.2 Protocolo HTTP

O protocolo HTTP é o responsável pela ligação das partes que unem a Web, atuando como um recipiente para todo o conteúdo que é trafegado entre *browsers* e servidores web, com exceção das comunicações que se baseiam em objetos distribuídos (DOS SANTOS et al., 2002 apud TEIXEIRA, 2004). Este protocolo tem como função realizar o acesso aos recursos armazenados no servidor web, atuando como um protocolo de nível de aplicação, que parte da premissa da existência de um serviço de transporte confiável, como o TCP (COMER, 2000 apud TEIXEIRA, 2004).

Além da transação HTTP ocorrer com base no conceito de requisição/resposta, o seu protocolo não guarda o estado do cliente, pois caso ocorra alguma falha, o cliente terá que solicitar novamente pela transação e executar todos os passos até completá-la (TEIXEIRA, 2004).

Segundo Teixeira (2004), apesar de sob a ótica do usuário a requisição de uma página HTML se traduzir em apenas um clique do mouse, por trás desta ação decorrem diversas requisições e respostas entre o *browser* e o servidor web, pois este comportamento decorre pelo fato de ser gerada uma requisição independente ao servidor para cada objeto contido numa página HTML, pois no protocolo HTTP 1.0, se traduz em uma nova conexão TCP para cada objeto requisitado, acarretando em uma sobrecarga desnecessária tanto no servidor quanto na rede.

Conforme Teixeira (2004) descreve, o protocolo HTTP é representado pelo padrão MIME, que também é usado no protocolo SMTP das mensagens de e-mail. Outro aspecto em relação ao HTTP é que normalmente o *browser* e o servidor web fazem a negociação dos dados que serão trocados, contribuindo para a sobrecarga do HTTP.

No que diz respeito às mensagens de requisição e resposta, o HTTP possui um padrão de mensagem para estes processos. Uma requisição possui basicamente uma linha informando qual ação deverá ser executada no servidor, a localização do documento no servidor e a versão do protocolo HTTP.

Em seguida vem uma ou mais linhas, que representam os parâmetros, como por exemplo, contendo as informações do servidor, os tipos de dados que ele suporta, como uma espécie de negociação e após segue-se o corpo da mensagem, que é opcional, utilizado para quando for necessário informar dados extras, como o método POST, por exemplo (TEIXEIRA, 2004).

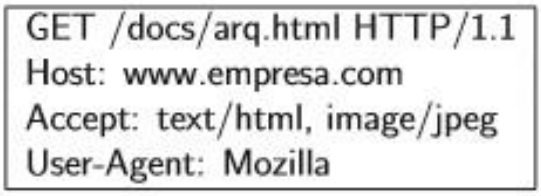


Figura 2.1: Exemplo de uma requisição HTTP
Fonte: Teixeira (2004)

De acordo com Teixeira (2004), para o processo de resposta do protocolo HTTP, é determinado que o padrão contenha uma linha de status, constituída da versão do protocolo, o código do status e a sua tradução.

Quanto à classe do código de status, referente ao resultado da execução solicitada pelo cliente, pode-se defini-la em cinco centenas (1xx à 5xx), sendo a primeira centena (1xx) com finalidade informativa, a segunda centena (2xx) indica o sucesso, a terceira centena (3xx) corresponde a questões de redirecionamento, a quarta centena (4xx) informa que houve erro no cliente e a quinta e última centena (5xx) informa que o erro foi no servidor.

Após esta primeira linha, a mensagem vem seguida de um ou mais campos que podem conter diversos campos informando as características do servidor e do objeto retornado ao cliente e por fim uma linha em branco precedida do corpo da mensagem que geralmente contém um documento HTML (TEIXEIRA, 2004).

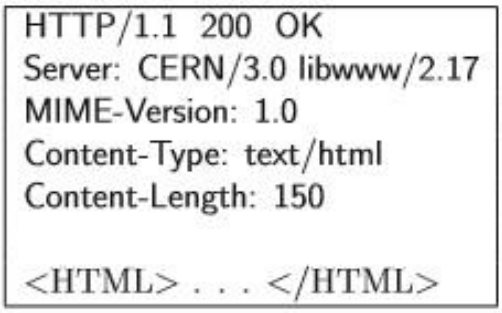


Figura 2.2: Exemplo de uma resposta HTTP
Fonte: Teixeira (2004)

Quanto aos métodos, Teixeira (2004) comenta que o protocolo HTTP suporta um conjunto, que pode ser utilizado pelo cliente, através de comandos enviados para o servidor web, no qual inicialmente a versão HTTP 1.0 definiu os métodos GET, HEAD e POST e posteriormente, na versão HTTP 1.1 passou a suportar os métodos OPTIONS, PUT, DELETE, TRACE e CONNECT. Abaixo, segue a definição de cada um destes métodos:

- **GET:** Solicita que seja retornado o recurso identificado pela URL.
- **HEAD:** Utilizado para a obtenção de informações de um determinado recurso sem que a haja a necessidade deste ser retornado ao cliente, sendo útil para validação de links, acessibilidade e data de última modificação.
- **POST:** Realiza o envio de informações extras do cliente para o servidor dentro do corpo da mensagem, como dados digitados em um formulário HTML, por exemplo.
- **OPTIONS:** Utilizado para receber os pré-requisitos sobre a comunicação e informações pertinentes ao recurso solicitado, sem necessitar iniciar sua recuperação.
- **PUT:** Permite criar ou modificar um recurso no servidor web.
- **DELETE:** Solicita a exclusão do recurso no servidor web identificado pela URL.
- **TRACE:** É usado para enviar uma mensagem de teste, do tipo *loopback*, ao servidor.
- **CONNECT:** Reservado para comunicação com servidores proxy.

No que se refere às versões do protocolo HTTP, segundo Teixeira (2004), a versão HTTP 1.0 surgiu junto com a Web em 1990, a qual apenas consistia numa maneira básica de recuperar dados pela internet, mas que apresentou algumas deficiências conforme a Web se expandia. Dentre estas deficiências, pode-se citar a sua exigência para estabelecer uma conexão TCP para cada objeto solicitado, o que faz sentido para o contexto da época, que lidava com documentos que continham basicamente texto.

Porém, atualmente, uma simples página HTML pode conter inúmeras imagens pequenas, por exemplo, acarretando sobrecarga tanto na rede quanto nos servidores. Já com o advento do protocolo HTTP 1.1, utiliza por padrão o mecanismo de conexões persistentes, podendo-se aproveitar a mesma conexão TCP para diversas transações HTTP, tornando-se mais eficiente (TEIXEIRA, 2004).

Entre as vantagens do HTTP 1.1, Teixeira (2004) destaca que o protocolo permite que diversas requisições sejam enviadas, sem a necessidade de esperar pelas respostas, sendo útil, por exemplo, na recuperação de imagens de uma página, especialmente nos ambientes onde existe uma alta latência para o estabelecimento de conexões TCP.

Além disso, houve uma disseminação no uso de caches na Web, na tentativa de diminuir a latência no acesso aos servidores, assim como o tráfego na rede durante transferências desnecessárias de arquivos. Devido a isso, foram incluídos nesta nova versão comandos específicos para manipular os caches tanto em servidores quanto em clientes (TEIXEIRA, 2004).

2.3 Arquitetura Cliente-Servidor

Atualmente, a Web sustenta a maior plataforma cliente-servidor, na qual é possível notar a aplicação dos protocolos TCP/IP e do conceito cliente-servidor em larga escala, na rede mundial. Pode-se definir um servidor web como um processo que fica constantemente esperando requisições vindas dos clientes, onde basicamente consiste em interpretar uma solicitação, validando-a e, em seguida, retornando o pedido realizado por tal cliente, que geralmente se trata de um documento HTML.

Este documento, por exemplo, pode estar armazenado fisicamente no sistema de arquivos do servidor (páginas estáticas) ou pode ser gerado de forma dinâmica por um programa ou um *script* solicitado pelo servidor (páginas dinâmicas). (YEAGER & MCGRATH, 1996 apud TEIXEIRA, 2004).

Segundo Yeager e McGrath (1996) contam, nos primórdios da Web, sua estrutura se tratava apenas de um sistema de hipertexto em escala mundial, através de milhares de documentos interligados, em sua maioria compostos por texto simples e outras mídias. Dessa forma, através dos *browsers*, servidores web, os documentos HTML, o protocolo HTTP e as URLs faziam com que o funcionamento da Web fosse possível.

A título de exemplificação, seu funcionamento básico ocorre quando um *browser* comunica-se com um servidor web após um usuário informar uma URL, onde neste processo, através do protocolo HTTP, o *browser* faz uma solicitação ao servidor. Assim que o servidor recebe esta requisição através de um *socket*, numa porta pré-definida, efetua uma conexão com o cliente. Logo após, o servidor busca em seu disco o arquivo que foi requisitado e envia ao cliente, fechando a conexão. Por último, com o recebimento da resposta pelo *browser*, faz um exame do conteúdo do arquivo enviado, e quando for o caso, faz a interpretação dos comando HTML e apresenta o conteúdo formatado ao usuário solicitante. (YEAGER & MCGRATH, 1996 apud TEIXEIRA, 2004).

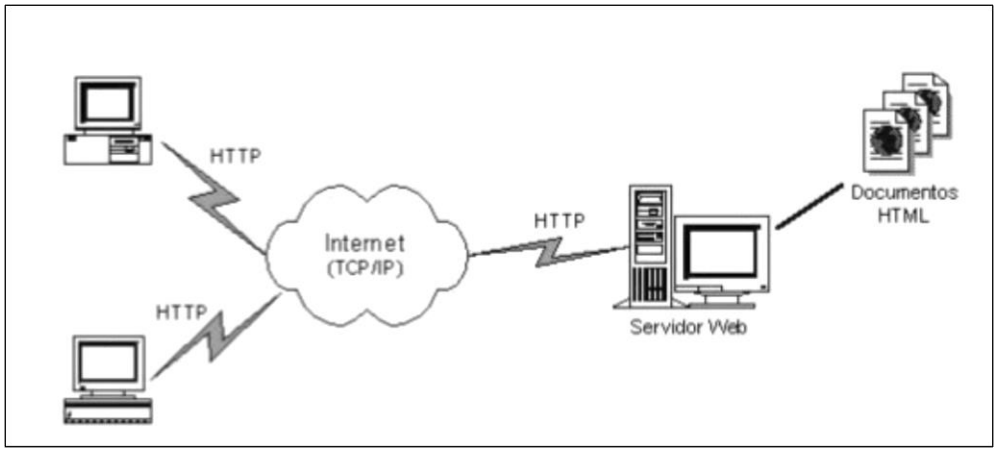


Figura 2.3: Interação cliente-servidor na Web
Fonte: Teixeira (2004)

Nesta situação, as conexões cliente-servidor são de duas camadas (*two-tier*), pois os *browsers* acessam uma infraestrutura de servidores de arquivos HTML com clientes considerados leves. Como os arquivos HTML estão basicamente armazenados fisicamente no sistema de arquivos do servidor web, esta fase inicial da Web apresenta-se com uma baixa funcionalidade e pouca interatividade com o usuário quando este realiza as suas solicitações. Até então, o processo ocorria através do conceito de páginas estáticas.

Através do protocolo CGI, criado ao final de 1995, a Web passou a receber mais interatividade, possibilitando que um *browser* conseguisse iniciar uma aplicação do lado servidor (YEAGER & MCGRATH, 1996 apud TEIXEIRA, 2004). Para garantir a independência de plataforma, as interações entre *browser* e servidor web continuam ocorrendo no formato HTML.

Na essência, o que o CGI faz principalmente, é realizar a transferência do pedido de execução ao programa correto, localizado no lado servidor, atuando como um tradutor do código HTML enviado pelo cliente e os requisitos de cada aplicação, podendo ser um servidor de e-mail ou FTP, um banco de dados, entre outros. As aplicações CGI possuem, como porta de entrada, os formulários HTML, os quais contêm os parâmetros informados pelo usuário através do *browser*.

Para fins de exemplo, numa transação HTTP com o uso de CGI, o programa servidor, após receber uma solicitação, processa e devolve, em formato HTML, o resultado ao módulo CGI, repassando ao cliente, pois, para o usuário, causa a impressão de ter acessado uma página estática no servidor, sendo que na realidade esta página foi

criada dinamicamente através de um processo (chamado de programa ou script CGI) iniciado por um servidor web.

Este processo não depende do servidor, pois, para o sistema operacional, sua execução decorre em um outro espaço de endereçamento e com escalonamento próprio. A definição que o protocolo CGI faz é no que diz respeito ao formato de informações que ocorrem nas interações do *browser* com o programa acionado, caracterizando com isto o modelo de páginas dinâmicas, que se difunde cada vez mais pela Web.

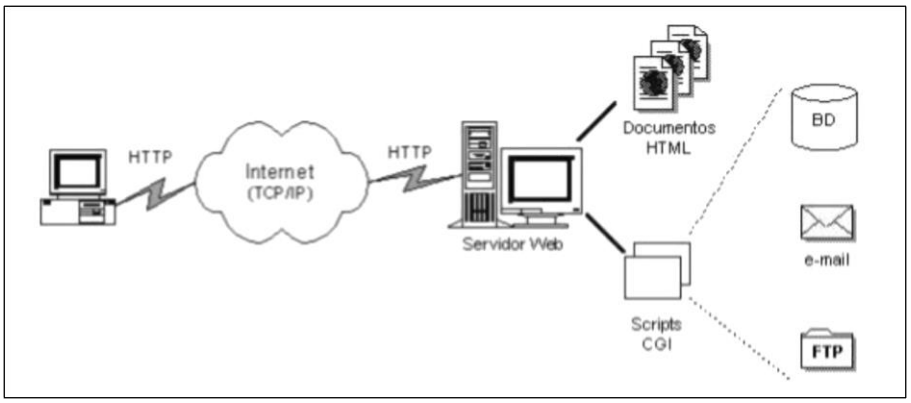


Figura 2.4: Interação cliente-servidor baseada em páginas dinâmicas
Fonte: Teixeira (2004)

A comunicação cliente-servidor neste caso são de três camadas (*three-tier*), sendo que a camada intermediária é formada pelo servidor web em conjunto com o módulo CGI e as camadas externas são representadas pelos *browsers* clientes e pelas aplicações que servem a objetivos específicos, como um banco de dados, FTP, E-mail, entre outros.

Apesar de atualmente o CGI ser comumente utilizado para a geração de páginas dinâmicas na Web para o suporte às aplicações deste contexto, o seu protocolo demanda um tempo maior de resposta, além de possivelmente sobrecarregar o servidor, pois exige um novo processo para tratar cada requisição realizada (ORFALIET al.,1999 apud TEIXEIRA, 2004).

Ao longo do tempo, apareceram algumas soluções como a ASP, *Cold Fusion*, HTML dinâmico, NSAPI, PHP, *servlets* Java, entre outros, que visam amenizar estas desvantagens do CGI, como na tentativa de compartilhar processos em memória entre invocações de serviços e introdução de uma maior interação no cliente, porém, algumas destas soluções pecam por não serem universais, pelo fato de serem proprietárias (TEIXEIRA, 2004).

2.4 Arquitetura de servidores web

Os servidores web são aplicações que devem disponibilizar tanto serviços quanto conteúdos para que as aplicações web possam realizar o acesso de forma distribuída e concorrente. Além do mais, estes servidores devem apresentar suporte no que diz respeito à autenticação, autorização e controle de acesso, filtro de dados na entrada e saída, segurança, alta performance e escalabilidade (DE OLIVEIRA JUNIOR, DE MATTOS FORTES).

Nesta seção serão abordados os tipos de arquitetura de servidores web, pois são estes que tratam as solicitações dos usuários, que dependendo do volume destas requisições, podem acarretar uma sobrecarga ao sistema. Uma das principais características de um servidor web é quanto ao fato de ficar constantemente aguardando por requisições dos clientes.

Enquanto aguarda pelas requisições, ocorrem de surgir inevitáveis atrasos, como por exemplo, a transmissão de dados pela rede o acesso ao disco do servidor, o escalonamento de processos no sistema operacional, entre outros. Tendo-se isto em vista, um servidor deve ser criado para que possa atender o máximo de solicitações que sejam possíveis, onde existem várias arquiteturas que propõem otimizar seu desempenho no que se refere ao seu nível de concorrência pelos clientes.

Quanto aos tipos de arquitetura, conforme Teixeira (2004) descreve, uma arquitetura de Servidor Iterativo, pode-se dizer é o tipo de servidor mais simples, pois consiste em aguardar as requisições dos clientes, tratando-as individualmente, em ordem de chegada e sem concorrência, que na prática serve apenas para fins didáticos, pois em operação não seria uma arquitetura eficiente.

Já a arquitetura de Processo por Requisição representa uma das formais mais utilizadas na projeção de servidores concorrentes, pois neste modelo existe um processo principal (processo “pai”) que através de uma porta inicialmente definida fica esperando por requisições. Quando surge uma nova requisição, este processo principal cria uma cópia de si (processo “filho”) e passa a ela a função de tratar a requisição e após volta ao seu laço de espera.

> “No UNIX, a criação dos processos filhos é feita por meio de uma chamada à *system call* **fork()** (Stevens et al., 2003), a qual gera um novo processo que é uma cópia do pai, possuindo o mesmo código executável e o mesmo conteúdo da memória, sendo-lhe passado um identificador da conexão de rede que foi estabelecida com o cliente. Este novo processo é executado e escalonado

> independentemente do processo pai e, ao terminar de atender a solicitação do cliente, encerra sua execução e deixa de existir.” (TEIXERA, 2004).

Após um dado momento, o servidor web possuirá diversas cópias suas que estarão sendo executadas em paralelo, limitadas somente pela máquina que as abriga, sendo que, devido a isso, o servidor determina um número de processos concorrentes, onde numa situação de ultrapassagem deste limite, provavelmente ocorrerá o bloqueio de sua execução, pois cada processo filho ocupa recursos da máquina, como memória e processamento, além do estabelecimento de uma conexão com o cliente. Portanto, alguns servidores web determinam uma limitação de processos filhos que podem executar paralelamente (YEAGER & MCGRATH, 1996 apud TEIXEIRA, 2004).

Em termos computacionais, a criação de um novo processo se mostra cara e, de certa forma, desnecessária, pois as transações web se tratam de operações simples e para a sua execução se dispensaria a necessidade de um processo exclusivo. Porém, em decorrência da simplicidade e naturalidade de sua implementação, especialmente em ambientes UNIX, se justifica a popularidade para a projeção de diversos tipos de servidores.

Na arquitetura *Pool* de Processos, se oferece uma solução na tentativa de aproveitar a simplicidade da arquitetura de Processos por Requisição, eliminando as desvantagens desta. Neste modelo, ao iniciar o servidor, cria-se um número mínimo de processos, chamado *process pool* (HU et al., 1997 apud TEIXEIRA, 2004) e existe um outro processo denominado *dispatcher*, que possui a finalidade de ficar constantemente aguardando as requisições dos clientes. Quando recebe uma solicitação, o *dispatcher* seleciona um dos processos do *pool* para atendê-lo, e na sequência, retorna ao seu estado de espera.

Dentre os processos do *pool* que realizam o tratamento da requisição, estes não são encerrados, ao invés disso, retornam ao seu estado ocioso, onde ficam prontos para atender novas solicitações convocadas pelo *dispatcher*. O Apache é um exemplo de servidor web que faz uso dessa arquitetura (APACHE SOFTWARE FOUNDATION, 2002 apud TEIXEIRA, 2004).

Neste modelo, observa-se a vantagem da eliminação na sobrecarga de criação de processos, mas possui a desvantagem de poder sobrecarregar o *dispatcher*, representando o gargalo no servidor, pois nele inicialmente se passam todas as requisições, além de ser um ponto crítico de falha.

Outro problema seria na determinação de um número ideal de processos *pool*, pois este número é relativo a carga que o servidor possa vir a receber, que caso seja alta, precisaria criar novos processos, apresentando a desvantagem do modelo de Processos por Requisição.

Outra arquitetura existente é a *Threads* por Requisição, tendo o seu princípio baseado na criação de uma nova *thread* para tratar cada nova requisição que chega, pois este modelo apresenta a vantagem de que *threads* consomem menos recursos de uma máquina do que um processo, pois as *threads* podem compartilhar o mesmo espaço de endereçamento, códigos e dados globais, além da mudança de contexto entre elas ocorrerem de forma mais rápida pelo fato de estarem dentro do mesmo processo.

Como os servidores web recebem tarefas que normalmente precisam aguardar por I/O, pois necessitam esperar a leitura dos dados no disco, além de terem de aguardar pela recepção e envio dos pacotes pela rede, o uso de *threads* para estes casos surgem como uma solução apropriada, pois enquanto uma *thread* aguarda o I/O, o controle é repassado à outra *thread*, que atende a uma requisição diferente, o que resulta na utilização mais eficiente dos recursos da máquina. Um exemplo de servidor que usa o modelo de *thread* por requisição é o PHTTPD (HU et al., 1998 apud TEIXEIRA, 2004).

Através do uso de *threads*, o servidor web apresenta um desempenho mais rápido, permitindo a execução de requisições em paralelo, mostrando-se como uma solução vantajosa na criação dos servidores atuais. Porém, apresenta como uma relativa desvantagem o fato da programação de *threads* serem mais complexas de se implementar do que processos, além de seu código, eventualmente, necessitar de uso de bibliotecas específicas de um determinado sistema operacional, afetando a sua portabilidade (YEAGER & MCGRATH, 1996 apud TEIXEIRA, 2004).

Por fim, outra arquitetura possível é a utilização de um *pool* de *threads*, criadas na inicialização do servidor web. Assim como ocorre com o *pool* de processos, a mesma ideia de implementação é realizada, pois desta maneira evita a ocorrência de sobrecarga para a criação de novas *threads* para novas requisições feitas ao servidor. O servidor JAWS (HU et al., 1998 apud TEIXEIRA, 2004) utiliza este modelo (TEIXEIRA, 2004).

Os servidores web mais populares são *Apache HTTP Server*, *HP Web Server Suite* e o *IBM HTTP Server*, mas dentre estes, o que é mais utilizado é o Apache pelo fato de não ser software proprietário, além da estabilidade que oferece devido ao

desenvolvimento de seus recursos ao longo dos anos. O Apache também se demostra como um servidor versátil, pois permite a instalação de vários módulos individuais que oferecem suporte ao processamento de dados enviados pelos clientes através de diversas tecnologias, entre elas o JSP, ASP, PHP, CGI e Perl (DE OLIVEIRA JUNIOR, DE MATTOS FORTES).

Quanto ao modelo de arquitetura de um servidor web, pode-se destacar um modelo apresentado por Menascé & Almeida (1998) apud Teixeira (2004), destacado na Figura 2.5, cujo modelo é constituído de seis centros de serviço. O primeiro representa um nó de retardo e simboliza os clientes, representando o tempo destes (*think time*), que determina o tempo entre a recepção de um arquivo e o início da próxima solicitação.

O segundo nó simboliza a rede local e o terceiro, um roteador, também modelado como um nó de retardo devido a sua baixa latência em comparação aos demais nós. Os links de saída e chegada do provedor de internet são representados nos nós 4 e 6 e o nó 5 simboliza o provedor, seus links para a internet e seus servidores web, numa visão macro. Na rede de filas, é representada pelas requisições HTTP em circulação.

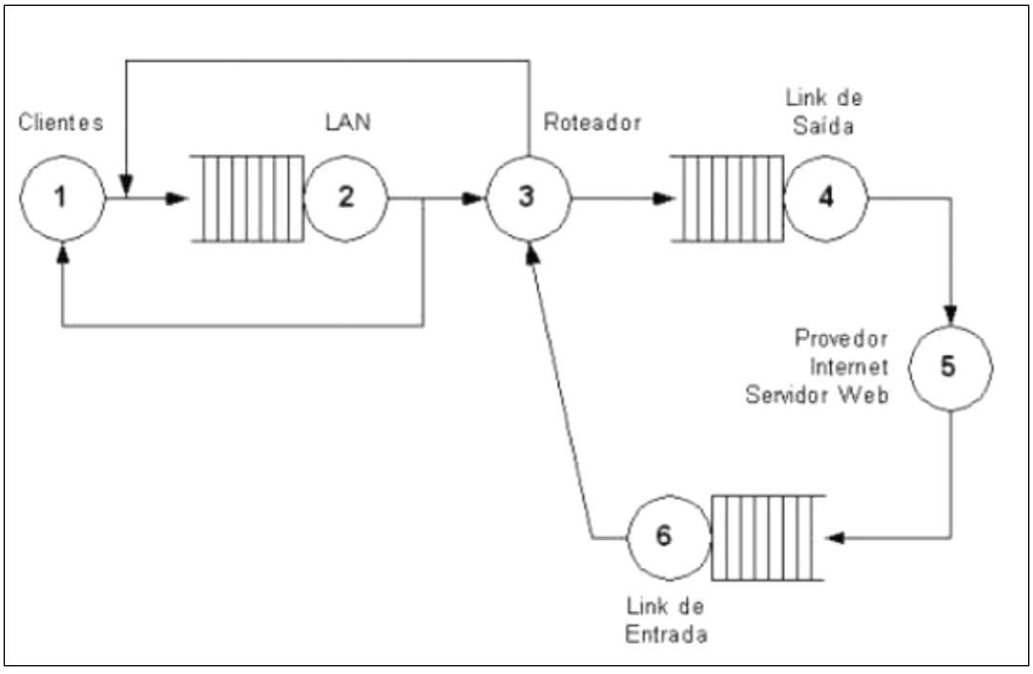


Figura 2.5: Modelo de rede de filas para a Web
Fonte: Teixeira (2004)

2.5 *Web Services*

Um *Web Service* (WS), conhecido também por solicitante ou consumidor, consiste em um tipo de aplicação para a Web, que decorre através do protocolo HTTP, caracterizando-se como uma aplicação distribuída, o que possibilita a sua execução em dispositivos variados, podendo estes serem computadores pessoais PCs, celulares, *tablet*, entre outros (PINTO, 2013).

Os WSs se tratam de aplicações auto descritivas, modulares, acessíveis através de uma URL, não dependem de plataformas de desenvolvimento, além de possibilitar comunicações entre aplicações sem a necessidade de interação humana. Entretanto, com estas características, os WSs mostram-se como uma solução que visa sanar o problema de quando se precisa integrar aplicações, atribuindo-se estas possibilidades ao fato desta aplicação basear-se em normas padronizadas, dentre elas: XML, SOAP, WSDL e UDDI.

No XML, encontra-se a metalinguagem de anotação, onde estão estabelecidas todas as demais normas nas quais os WSs se baseiam (W3C, 1997; RAMALHO & HENRIQUE, 2002 apud LOPES & RAMALHO, 2004). O protocolo SOAP é caracterizado por ser a linguagem de anotação onde é descrito o protocolo de comunicação, responsável pela troca de mensagens entre os WSs (SOA apud LOPES & RAMALHO, 2004), no qual uma mensagem SOAP trata-se de um documento XML. Já o WSDL se trata da linguagem de anotação definida em XML, que objetiva realizar a descrição da API de um WS (WDS, NEWCOMER, 2002 apud LOPES & RAMALHO, 2004). Por fim, o UDDI, que também é uma linguagem de anotação definida em XML, responsável por conter a meta-informação de um WS, onde diversos registros UDDI ficam contidos em repositórios, permitindo que uma aplicação cliente possa pesquisar e localizar um serviço através de uma API (UDD apud LOPES & RAMALHO, 2004).

Um WS possui quatro estados distintos em seu ciclo de vida, sendo eles: Publicação, Descoberta, Descrição e Invocação. Abaixo as suas definições:

- **Publicação:** É o processo (que pode ser opcional), em que o fornecedor do WS apresenta publicamente o seu serviço, onde fica registrado no repositório de WS (UDDI).

- **Descoberta:** É o processo (que pode ser opcional), no qual uma aplicação cliente descobre a existência do WS solicitado, onde o pesquisa em um repositório UDDI.

- **Descrição:** É o processo em que o WS apresenta a sua API (documento WSDL), disponibilizando a sua interface com todas as suas funcionalidades descritas para a aplicação cliente, além dos tipos de mensagens que acordam com o uso delas.

- **Invocação:** É o processo no qual o cliente e o servidor se comunicam, pelo envio de mensagens de entrada e de eventuais recepções de mensagens de saída.

Quanto aos passos seguidos pelo WS em seu ciclo de vida, inicia-se com o fornecedor realizando a construção do serviço, utilizando a linguagem de programação que entender. Após, é especificada a interface, ou assinatura, do serviço definido em WSDL, sendo que depois de finalizados estes passos anteriores, o fornecedor realiza o registro no UDDI. Em seguida, a aplicação cliente, ou utilizador, faz sua pesquisa no repositório UDDI e localiza o serviço, e por fim, a aplicação cliente estabelece a comunicação com o WS, através de mensagens SOAP (LOPES & RAMALHO, 2004).

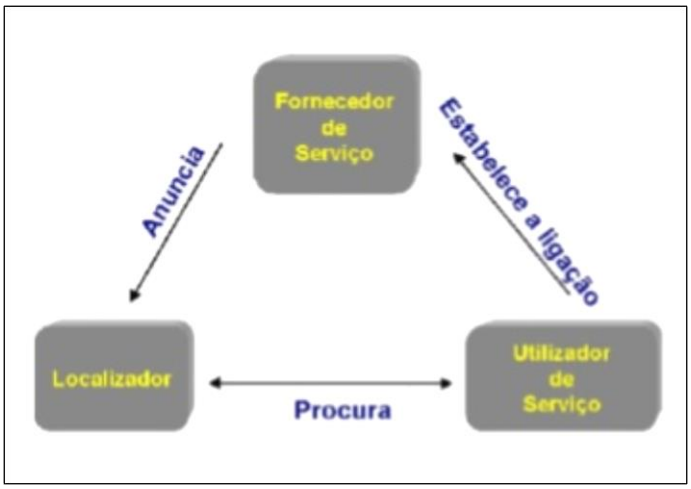


Figura 2.6: Ciclo de vida de um Web Service
Fonte: Lopes & Ramalho (2004)

A seguir será descrito os principais protocolos mais utilizados em WSs, sendo eles o SOAP e o REST.

2.5.1 Protocolo SOAP

A primeira aparição pública do SOAP foi em 1998, derivado de um trabalho da Microsoft que consistia no desenvolvimento de um sistema distribuído com base em XML, pois o objetivo consistia em que aplicações pudessem se comunicar através de chamadas de procedimentos remotos (RPC) com base em XML para a definição de um

protocolo. Neste trabalho, inicialmente estiveram envolvidos a IBM, DevelopMentor e Useland (GRAHAM et al, 2004 p. 112 apud TAVARES, 2012).

Nos anos 2000, com sua versão 1.1, o protocolo foi remetido para aprovação da W3C, onde até o momento, o nome do protocolo simbolizava a abreviação de *Simple Object Access Protocol*, que teve esta representação abandonada nas versões futuras, visto que não há a necessidade de objetos serem usados na sua implementação (SUDA, 2003 apud TAVARES, 2012), passando a ser um protocolo recomendado pela W3C a partir da versão 1.2 (TAVARES, 2012).

O SOAP se caracteriza como um protocolo para a troca de informações estruturadas, baseadas em XML, possuindo orientação de arquitetura voltada à serviço SOA, além de ser um protocolo onde documentos são definidos como mensagens.

Nos WSs com base em SOAP, esta é a infraestrutura menos vista, onde, por exemplo, numa determinada situação como numa troca de mensagem de pedido/resposta, a biblioteca SOAP do cliente manda uma mensagem SOAP solicitando o serviço e a biblioteca SOAP do WS manda outra mensagem SOAP respondendo ao serviço, ou seja, SOAP se trata do envelope das mensagens, possuindo as regras de codificação, estrutura de comunicação, além de tratamento de erros.

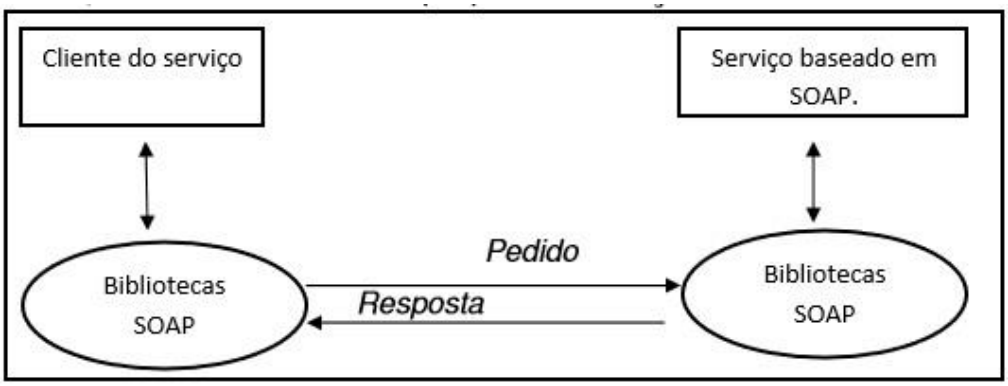


Figura 2.7: Arquitetura de um típico WS baseado em SOAP.
Fonte: Pinto (2013)

O SOAP mostra-se versátil no que diz respeito aos mecanismos de transporte, pois possibilita sua integração com vários protocolos para transmitir as suas mensagens, como o HTTP, SMTP e MQ*Series*, mas, o protocolo HTTP, junto ao seu método POST é o mais utilizado nas suas implementações. Apesar do uso de uma mesma URL para o envio de todas as mensagens não ser um requisito na sua especificação, é uma estratégia comum entre os promotores do SOAP (PINTO, 2013).

Quanto ao modelo de mensagem SOAP, define-se que deva ser estruturada e estabelecidas as regras para o seu processamento. Seu modelo se trata de um documento XML, no qual o elemento raiz é o envelope, podendo possuir até dois filhos, sendo eles o cabeçalho e o corpo (CURBERA et al, 2002 apud TAVARES, 2012). O cabeçalho SOAP é uma parte opcional, onde eles possibilitam a criação de mecanismos de extensão para o envio de informações que não constituem a mensagem criada pela aplicação. No corpo SOAP fica contida a mensagem que a aplicação transmitirá no envio da mensagem, sendo que o nodo que processará esta mensagem procurará no corpo SOAP o conteúdo a ser processado.

No que tange as mensagens com finalidade de invocar WSs, estes documentos podem ser definidos como RPC e documentada. Uma mensagem do tipo RPC, apresenta-se de forma a mostrar que a estrutura fará a invocação de uma função, através da definição do nome da função como elemento único no corpo da mensagem. Este elemento poderá conter filhos que possam representar os seus parâmetros e devido a isto, o seu posicionamento deve estar na ordem esperada pelo procedimento remoto. Quanto ao retorno, na forma de XML, possui um nodo raiz único pelo qual o retorno da função é representado (BOX et al, 2000 apud TAVARES, 2012).

Já numa mensagem codificada do tipo documentada, esta não possui nenhuma restrição no que se refere ao formato da mensagem, ou seja, significa que dentro do documento XML a mensagem trafegará livremente, sem qualquer tipo de restrição quanto a forma e implementação, implicando no acordo que deve existir entre as aplicações de origem e destino no que diz respeito à forma de interpretar as informações que devem ser transmitidas (ROTHAUG, 2004 apud TAVARES, 2012).

2.5.2 Protocolo REST

O protocolo REST (Transferência de Estado Representacional), consiste em uma técnica de engenharia de software para sistemas hipermídia distribuídos como a Web. A origem do termo surgiu em 2000, na tese de doutorado sobre a Web (FIELDING, 2000 apud PINTO, 2013), descrita por Roy Fielding, sendo ele um dos autores que definiram as especificações sobre o protocolo HTTP.

Inicialmente, o termo REST fazia referência aos princípios da arquitetura de sistemas de informações distribuídos, num primeiro momento chamada de "*HTTP Object Model*", sendo atualmente utilizada, de forma macro, para representar qualquer interface

web que faz uso do XML e HTTP, assim como o protocolo que baseia-se em um padrão de troca de mensagem em um WS, como o SOAP, possibilitando o projeto de sistemas web, de acordo com a arquitetura do protocolo REST descrita por Fielding (PINTO, 2013).

A utilização do REST se deu de forma ampla, a fim de demonstrar a alta escalabilidade do protocolo HTTP, ao passo que ambos acabam sendo tratados como sinônimos, pois de alguma forma, todo recurso disponibilidade por HTTP aplica alguma recomendação REST.

A idealização do REST visou fazer o uso do já consolidado protocolo HTTP para realizar a transferência de dados entre computadores ao invés de fazer uso de um protocolo que atuasse sobre o HTTP para realizar a transferência de mensagens, pois uma aplicação que seja criada baseando-se nos princípios do REST, poderá enviar e receber requisições fazendo o uso do HTTP ao invés de fazer uso de soluções mais complexas como CORBA e SOAP (POTTI, 2011 apud TAVARES, 2012).

Para o entendimento mais claro do REST, é necessário que se compreenda alguns de seus conceitos, entre eles existe o recurso, no qual se trata da chave para a abstração do REST, consistindo em qualquer informação que possa ser nomeada como por exemplo, documentos, imagens, serviço do tempo, entre outros. Um recurso simboliza um mapa conceitual para um conjunto de entidades, entretanto, um recurso não representa uma entidade que corresponda a um mapeamento em qualquer espaço de tempo (FIELDING & TAYLOR, 2002 apud TAVARES, 2012).

Fazendo-se uma analogia, pode-se comparar com um versionamento de arquivos na implementação de um software por exemplo, cujos arquivos fazem parte de uma mapeamento, que ao longo do tempo, possuem seus valores alterados. Portanto, pode-se ter um arquivo na versão 1.2 e na versão 1.5 deste software, pois mesmo que não seja feita nenhuma alteração neste arquivo, ele é representado por dois recursos diferentes, apesar de fazerem referência ao mesmo arquivo.

Outra definição que pode-se atribuir à recurso no REST é como qualquer coisa na Web que possa ser acessado e processado por algum cliente para alguma tarefa (WEBBER, PARASTATIDIS & ROBINSON, 2010 apud TAVARES, 2012), pois mesmo no mundo real, os objetos podem ser representados na Web sob a forma de recursos. Isso pode ser feito quando se cria uma abstração das informações mais

relevantes e fazer a inserção deste resultado no meio digital, possibilitando que materiais de escritório ou até pessoas possam ser representados como recursos, apenas fazendo com que suas informações mais importantes fiquem disponibilizadas na rede.

Outro conceito a ser definido no REST é um identificador uniforme, pois todo o recurso precisa de uma identificação e de um endereçamento em uma rede para que um recurso possa ser acessado e manipulado.

Portanto, a Web definiu como tecnologia padrão para esse propósito a URI, que possibilita a identificação de um recurso e também para torná-lo endereçável, permitindo com que protocolos de aplicações como o HTTP possam acessá-lo, pois a URI faz a distinção de um recurso em relação a qualquer outro na Web, além deste recurso poder ser identificado por mais de uma URI (WEBBER, PARASTATIDIS & ROBINSON, 2010 apud TAVARES, 2012).

Um conceito importante a ser definido no REST se trata da representação, que se traduz em qualquer informação útil que esteja sobre o estado de um recurso (RICHARDSON & RUBY, 2007, TAVARES, 2012), pois o REST não possui qualquer tipo de restrição no que se refere a representação física dos recursos. Um recurso tem seu acesso determinado pela forma como ele é representado, pois a Web consiste na troca de representação de recursos e não diretamente pela troca destes.

Com essa separação, observa-se o baixo acoplamento entre a aplicação que disponibiliza a representação do recurso e o sistema cliente, pois através desta característica, observa-se também a escalabilidade, possibilitando que representações possam ser replicadas ou salvas em caches (WEBBER, PARASTATIDIS e ROBINSON, 2010 apud TAVARES, 2012).

Uma outra definição técnica para a representação é como se fosse a serialização de um recurso através da escolha da sintaxe (FERREIRA, 2009 apud TAVARES, 2012). Um serviço pode oferecer a um recurso mais de um tipo de serialização, pois nesta situação o formato desejado (XML, XHTML, JSON, entre outros) deve ser escolhido pelo cliente, na qual a resposta para a requisição realizada apresenta o formato usado através do cabeçalho *Accept* do HTTP (TAVARES, 2012).

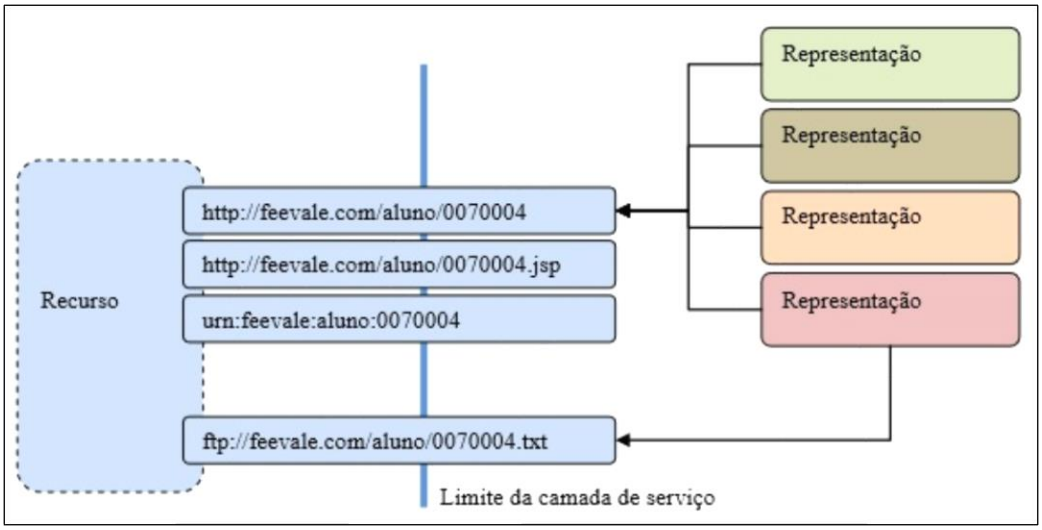


Figura 2.8: Representação Recurso x URI
Fonte: Tavares (2012)

Neste capitulo foram apresentadas as principais tecnologias utilizadas para que seja possível projetar um servidor de dados, em especial do tipo web, inserida no seu contexto de atuação, que será através da internet.

Foram abordados protocolos fundamentais como o TCP/IP, considerado um dos protocolos mais eficientes para o uso no tráfego de informações na web, tratando-se de um padrão mundial, pois oferece uma série de vantagens como ser flexível quanto a arquitetura de rede, é um padrão aberto, além de possuir controles em sua arquitetura de camadas que provem a integridade e confiabilidade dos dados.

Analisou-se também o protocolo HTTP, que basicamente é o protocolo responsável por interligar a web, possuindo como função o encapsulamento do conteúdo trocado entre *browsers* e servidores web, com base no conceito de requisição/resposta entre cliente e servidor. Pôde-se conhecer a estrutura deste protocolo usado no processo de comunicação, além de seus métodos de atuação.

Verificou-se como é e como funciona uma arquitetura cliente-servidor, sendo apresentado em maiores detalhes o funcionamento desta interação com páginas estáticas ao solicitar documentos HTML e páginas dinâmicas com o mesmo propósito, mas com maior interatividade através do protocolo CGI, possibilitando o suporte para as requisições/respostas entre clientes e servidores.

Se pôde averiguar os conceitos, explicando sobre as atribuições que o servidor tem no sentido de oferecer suporte à autenticação, acesso, segurança, escalabilidade, performance, entre outros. Além disso, foram detalhados alguns modelos de arquiteturas

de servidores web, todos eles abordando determinados tratamentos no que tange a dinâmica de múltiplas requisições e respostas para o funcionamento eficiente do servidor.

Foi estudado também sobre os fundamentos do WS, em que apresenta suas vantagens quanto a integração de aplicações, independência de plataforma e da interação humana e demais vantagens, possuindo sua base no formato XML. Junto a isso, foi abrangida a exploração dos protocolos SOAP e Rest, dos quais verificou-se suas principais características quanto a suas estruturas e modos de funcionamento.

3. ANÁLISE DA PLATAFORMA DO DISPOSITIVO MÓVEL

A plataforma que será analisada será o *Android*, da qual serão descritas as suas principais características.

3.1 Surgimento do *Android*

A plataforma *Android* consiste em um conjunto de softwares para dispositivos móveis, pois é composto por um sistema operacional e aplicativos fundamentais. Se trata de um projeto de código aberto, que foi concebido para o propósito de ser utilizado para aparelhos com diversas especificações (OHA, 2009b apud MARTINS, 2009).

Dentro de um contexto globalizado, este é o objetivo do OHA, que se trata de um grupo formado pelas empresas que lideram o mercado de tecnologia móvel, na qual procura determinar uma plataforma única e aberta para celulares, para atingirem a satisfação dos consumidores com o produto final, além de oferecer para o desenvolvimento de aplicações corporativas uma plataforma aberta e flexível. (LECHETA, 2010 apud PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]).

Dentre as tecnologias suportadas pela plataforma estão o *touchscreen* (tela sensível ao toque) e GPS, permitindo a localização pelos parâmetros de latitude e longitude. O *Android* apresenta uma infinidade de aplicações nativas como navegador, galeria de imagens, tocador de música, gerenciador de contatos, calculadora, entre outros (OHA, 2009c apud MARTINS, 2009). Assim como seus concorrentes, esta plataforma possibilita a criação de aplicações por terceiros (ARIMA, 2009a apud MARTINS, 2009).

Em relação ao mercado de dispositivos móveis, para muitas empresas, o *Android* foi o responsável por esta recuperação no setor, pois suas plataformas estavam deixando de serem utilizadas, além da queda na venda de seus aparelhos (CANALYS, 2010 apud UZEJKA, 2011).

Devido ao fato de ser uma plataforma aberta e de customização simples pelos fabricantes, o *Android* permitiu algo inédito, com a padronização de um sistema para diversos aparelhos, de várias marcas, pois a plataforma conseguiu apresentar um desempenho satisfatório, conseguindo ter uma participação significativa no mercado em pouco tempo (UZEJKA, 2011), pois, para as empresas, apresenta uma fácil customização, além de hardware acessível e barato (PEREIRA, 2009 apud PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]).

Outro ponto a ser considerado para a ascensão do *Android* é devido ao seu surgimento como uma opção atraente para os desenvolvedores que procuravam por uma padronização maior neste mercado tão segmentado (UZEJKA, 2011), além de conseguir propiciar todas as expectativas que espera-se em uma plataforma, como o fato de ser livre, confiável, robusta, código aberto e de uso facilitado. Por fim, atende as necessidades dos consumidores oferecendo aplicações ricas, interativas e integradas (PEREIRA, 2009 apud PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]).

3.2 Arquitetura do *Android*

A plataforma *Android* se trata de um sistema operacional que é executado sobre o *kernel* 2.6 do Linux, além de ser baseado em Java. Consiste em um sistema que possui muitos recursos, além de ser considerado leve. No que se refere aos seus aplicativos, estes são desenvolvidos também do uso da linguagem Java, oferecendo um alto nível de portabilidade. O paradigma é com base na orientação à objeto, baseados em XML, sendo layout de interface do usuário (DIMARZIO 2008 apud PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]).

Entretanto, para WALL(2008) apud UZEJKA(2011), o *Android* não é considerado somente um sistema operacional, trata-se de um conjunto de bibliotecas e ferramentas que estão acopladas a um *kernel* do Linux, que forma um sistema mais completo e robusto. Tais ferramentas comunicam-se entre si através de um formato de pilha, por meio das camadas inferiores oferecendo recursos às camadas superiores, com inspiração no conhecido modelo OSI, usado em redes de computadores. Na Figura 3.1 pode-se observar a composição da arquitetura do *Android*:

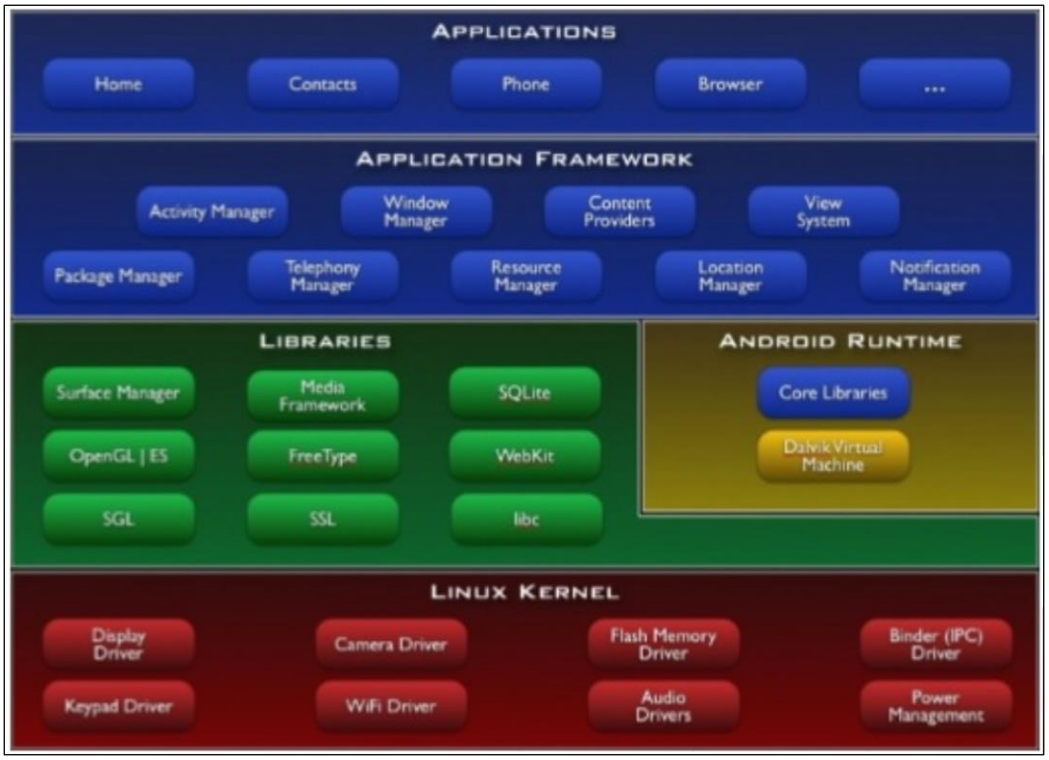


Figura 3.1: Arquitetura do Android
Fonte: GOOGLE (2010) apud MACK (2010)

Linux Kernel:

Na constituição de seu núcleo, encontra-se um *kernel* do Linux, na versão 2.6, no qual possui algumas alterações. Nele realiza-se o controle do gerenciamento da memória, dos processos e do sistema de arquivos. Juntamente à ele, estão os drivers de mais baixo nível, responsáveis pelo controle da tela, do teclado, dos dispositivos de rede, gerenciamento de rede, entre outros, servindo como base sobre a construção da plataforma. (UZEJKA, 2011).

> "O kernel também funciona como uma camada de abstração entre o hardware do dispositivo e o resto do conjunto de softwares que são desenvolvidos em paralelo (GOOGLE, 2007)." (PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]).

Libraries:

Na camada imediatamente acima, ficam disponíveis as bibliotecas escritas em C e C++, que respondem pelos gráficos, através do *Surface Manager* e *Open*GL, multimídia, banco de dados, por meio do SQ*Lite*, suporte a *browsers* pelo *WebKit*, além gerenciamento de fontes e camadas de rede.

Estas bibliotecas oferecem serviços para a camada acima, podendo ser chamada de framework de aplicações. (UZEJKA, 2011). Esta coleção de bibliotecas, utilizadas por várias partes do sistema, são disponibilizadas aos desenvolvedores, através da camada superior. (MARTINS, 2009).

Android Runtime:

No *Android*, cada aplicação executa em seu próprio processo, onde cada processo se trata de uma instancia da máquina virtual *Dalvik*, concebida para possibilitar que um dispositivo possa rodar, de forma eficiente, múltiplas maquinas virtuais (GOOGLE, 2007 apud PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]). Da mesma forma como diversos componentes do *Android*, esta máquina virtual foi otimizada especificamente para dispositivos móveis. (MARTINS, 2009).

Quanto a execução dos arquivos, é realizado pelo formato .dex, no qual foram otimizados para consumir o mínimo de memória possível, sendo a criação destes arquivos feita por um compilador Java, convertendo o resultado para o formato .dex (GOOGLE, 2007 apud PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]).

O baixo consumo e melhor desempenho da memória é possível devido a *Dalvik* ser baseada em registradores e por não utilizar o *bytecode* Java, pois o código binário usado contém um formato próprio, sendo gerado a partir do *bytecode* Java, contendo algumas simplificações e compactações (WALL, 2010 apud UZEJKA, 2011).

Application Framework:

A arquitetura deste framework foi concebida para facilitar o reuso dos componentes, pois desta forma, qualquer desenvolvedor que criar um aplicativo, poderá disponibilizar suas funcionalidades, possibilitando que outros programas possam utiliza-las (PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]). Segundo a Google (2007), "vale lembrar que o desenvolvedor tem acesso total à mesma estrutura de API's usada nos aplicativos centrais, podendo, desta forma, aproveitá-las conforme achar".

Applications:

Nesta camada estão inclusos os programas que realizam o gerenciamento das principais funções do telefone, como alocação de recursos, aplicações de telefone, mudança entre processos e programas, entre outros (MACK, 2010), assim como encontra-

se as API's que oferecem suporte ao desenvolvimento, facilitando o acesso ao sistema de localização, de telefonia, de notificação e demais serviços, além de oferecer a estrutura dos programas *Android*, através do gerenciamento do sistema de *views*, os *content providers*, as *activities* e o sistema de janelas. (UZEJKA, 2011).

Permite o acesso total pelos desenvolvedores para a extração das vantagens das capacidades do hardware do dispositivo, execução de serviços em segundo plano, entre outros. Como foi planejada para facilitar a reutilização de componentes, possibilita a criação de ferramentas mais elaboradas a partir de ferramentas mais básicas (MACK, 2010).

Na última camada da pilha, ficam o conjunto de aplicativos para cliente de e-mail, calendário, mapas, navegador e as principais funções para chamadas no telefone. É neste segmento que o usuário faz a interação com as interfaces dos aplicativos, ficando as camadas abaixo liberadas somente para desenvolvedores e fabricantes de hardware (MACK, 2010). Além da disponibilização de todo o software básico necessário para a boa utilização do dispositivo móvel, o *Android* possibilita que seus softwares possam ser substituídos caso o usuário prefira, sem que haja comprometimento do sistema.

Também permite a instalação de vários outros softwares, acrescentando ao aparelho todo o tipo de funcionalidade (UZEJKA, 2011). Seus aplicativos são escritos em Java, pois, depois de compilado, é empacotado em conjunto com os demais recursos que a aplicação utiliza em um arquivo de sufixo .apk, sendo este o meio de distribuição para os usuários instalarem as aplicações em seus dispositivos.

Normalmente, cada aplicativo é executado em seu próprio processo, onde cada processo tem sua máquina virtual, pois, para cada aplicativo é atribuída uma identificação exclusiva de usuário Linux, além das permissões serem dadas de maneira que os arquivos estejam visíveis somente para a aplicação dona. (MARTINS, 2009).

Em relação à prioridade de execução, os aplicativos de terceiros são executados na mesma prioridade das aplicações que rodam junto ao núcleo do sistema (CINDRAL, 2011 apud PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]), permitindo flexibilidade para colocar e executar as aplicações. Outro aspecto relevante é a permissão ao recurso de acesso a qualquer parte que o sistema operacional possa acessar, facilitando quando um desenvolvedor deseja criar uma aplicação que faça uso de recursos como de

discagem ou GPS interno, por exemplo (PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]).

Um aplicativo não possui um ponto de entrada único, sendo estas construídas através do uso de componentes que são instanciados de acordo com a necessidade do momento, sendo estes componentes definidos como atividades (*activities*), serviços (*services*), provedores de conteúdo (*content providers*) e receptores de *broadcast* (*broadcast providers*) (MARTINS, 2009).

Segurança no *Android*

De acordo com *Android Open Source Project* (2012a) apud BRAGA; DO NASCIMENTO; RODRIGUES (2012), o *Android* foi projetado para que a segurança pudesse ter mais independência com relação aos desenvolvedores, pois sua arquitetura de segurança possibilita que sejam colocados controles de forma transparente ao desenvolvedor, existindo uma alta abstração neste processo.

Consegue-se isto através do confinamento de aplicações, por meio de esquemas de permissões, tanto no sistema de arquivos através de chamadas da API, que por padrão, prioriza a segurança e por meio do mecanismo de IPC, que tais permissões são aplicadas para fornecer acesso entre os variados componentes.

A segurança à nível de dispositivos de hardware, assim como no Linux, o *Android* suporta várias configurações, como *smartphones*, *tablets*, *e-readers* entre outros, sendo que cada um destes dispositivos contém implementações de segurança em hardware, como por exemplo o ARM (*Advanced RISC Machine*) e o *TrustZone*, na qual estas capacidades são utilizadas pelo sistema.

À nível de sistema operacional, com base no *kernel* do Linux, a interface para a utilização do dispositivo é provida pelo núcleo do *Android*, por onde todos os recursos são acessados pela mediação do sistema operacional, ficando limitado aos seus controles de segurança.

Quanto à segurança no ambiente de execução de aplicações, a maior parte dos aplicativos da plataforma são implementados em Java, executando sobre a máquina virtual *Dalvik*. Mesmo assim, essas aplicações, em conjunto com outros serviços disponibilizados pelas bibliotecas e pelo *Android*, rodando código nativo, executam pelo *sandbox* de aplicação, que consiste em um ambiente isolado, sendo que este isolamento

limita as permissões de acesso da aplicação onde ela está sendo executada, ao sistema, aos recursos do sistema, aos dados de outras aplicações e a outras aplicações que estão em execução.

A arquitetura tem sua segurança baseada nos mecanismos de segurança usados pelo *kernel* do Linux e na disponibilidade de uma comunicação inter-processo segura, pois todo código de aplicação, inclusive os que rodam código nativo, estão limitados pelo *sandbox* de aplicação, que foi desenvolvido através de um modelo que isola os processos com base em usuários, no qual é aplicado pelo *kernel*. (BRAGA; DO NASCIMENTO; RODRIGUES, 2012)

3.3 Componentes do *Android*

Como citado anteriormente, uma aplicação *Android* é criada com base em quatro tipos básicos de componentes (LOMBARDO, 2009 apud UZEJKA, 2001), sendo eles:

Activities:

Este tipo representa uma interface visual que possibilita aos usuários a escolha de uma determinada tarefa a ser desempenhada, onde, por meio da composição destas atividades, que atuam de forma independente, pode-se criar de forma consistente uma interface ao usuário, pois cada atividade é designada como uma subclasse da classe base de atividade, permitindo uma vasta diversidade de atividades dentro das aplicações.

Uma aplicação pode ser construída a partir de uma atividade única ou através de um conjunto de atividades, podendo-se, por exemplo, ir de uma atividade para outra onde a atual atividade tenha sido iniciada em uma outra atividade. Além do mais, em um aplicativo é aplicada uma janela padrão que é desenhada por uma atividade, podendo estas janelas preencher a tela, flutuar sobre outras janelas ou interagir com elas (MACK, 2010).

Se tratam de componentes executáveis ou reutilizáveis criados pelo sistema operacional ou pelo usuário, tendo como objetivo efetuar a interação com o utilizador ou com outras atividades ou serviços, através da troca de informações ou serviços pelas *Intents*. Para fins de gerencia de memória, o sistema operacional pode encerrar este componente em caso de inatividade. Pode-se dizer que uma aplicação *Android* é formada em sua maior parte por *Activities* (UZEJKA, 2011).

Quanto ao ciclo de vida de uma atividade, pode-se saber se uma atividade está ativa ou inativa, se está visível ou invisível ao usuário, sendo formado basicamente por três estados.

O primeiro deles é quando a atividade está ativa ou rodando (*active or running*), que consiste quando a atividade está em primeiro plano, aguardando pelas ações do usuário, posicionando-se no topo da pilha das tarefas a serem executadas.

O outro estado significa quando a atividade está pausada (*paused*), pois representa que saiu do foco do usuário, apesar de continuar visível a ele, pois fica rodando em segundo plano devido a outra atividade estar cobrindo ela parcialmente, mas permanece residente no sistema com todas as suas informações.

Por fim, a atividade pode estar parada (*stopped*), pois significa que uma outra atividade a cobriu totalmente, ficando invisível ao usuário, mas que continua rodando seus estados e informações enquanto o sistema não a elimina.

O sistema notifica cada um destes estados, captável através da aplicação, permitindo a um desenvolvedor definir um comportamento ou funcionalidade para cada ação da atividade (MACK, 2010). Através da sobrescrita dos métodos da classe *Activity* (*on*[Evento]*()*), no momento adequado, o *Android* se encarregará de executá-los (BURNETT, 2009 apud PACHECO JÚNIOR; DE OLIVEIRA CASTRO, [s.d.]). A Figura 3.2 mostra o ciclo de vida de uma atividade:

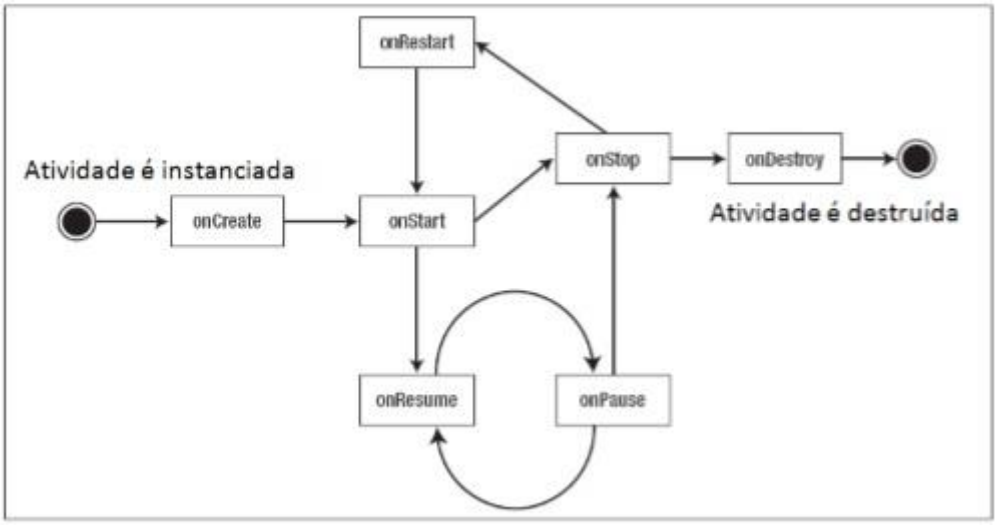


Figura 3.2: Ciclo de vida de uma atividade
Fonte: Martins (2009)

Services:

Os serviços se caracterizam por não possuírem uma interface visual, cujo propósito é realizar processamentos em segundo plano (MARTINS, 2009). Geralmente são usados quando se precisa prover um serviço por um tempo longo, sem que se tenha problemas de encerramento, ou quando se tratam de componente que não exigem uma interface para o usuário. Por exemplo, a realização de um download pode ser considerado um serviço, visto que é uma tarefa que executa em segundo plano e que não pode ser encerrada antes de completada, para isto, utiliza-se um serviço (UZEJKA, 2011).

Outro exemplo de serviço se trata do *player* de música, pois como as músicas são executadas de uma lista, a aplicação foi criada em meio a múltiplas atividades que possibilitam ao usuário escolher músicas e executa-las. Em contrapartida, a execução da música por si só não é uma atividade, sendo considerada um serviço, pois a *thread* principal do processo da aplicação é onde são executados os serviços.

Os serviços, assim como as atividades, também tem um ciclo de vida, com a diferença que os usuários não o visualiza, possuindo somente os estados *running* e *stopped*, que são iniciados pela chamada de *ContenstartService()* ou parado pela chamada de *ContenstopService()* (MACK, 2010).

Broadcast Receivers:

Este componente caracteriza-se por ser muito simples e seu funcionamento ocorre através da reação da recepção da transmissão de anúncios, pois muitos *broadcasts receivers* começam por meio do sistema, seja através de anúncios de bateria fraca ou mudança do idioma. As transmissões também podem ser iniciadas por vários aplicativos, pois isso possibilita que as aplicações consigam se comunicar com outras aplicações (MACK, 2010).

Pode também ser entendido como um gerenciador de notificações e eventos, derivados de aplicativos ou do sistema. Quando uma aplicação precisa comunicar-se com alguns módulos do dispositivo, podendo ser aplicações, como as informações de contato, ou o próprio sistema, através da sinalização de uma chamada perdida, deve conter um componente desse tipo. A comunicação ocorre por meio de uma *intent* (UZEJKA, 2011), que é o nome do objeto que possui uma mensagem com a ação que deseja-se executar,

como por exemplo, no início de uma atividade, é preciso enviar uma *intent* para que o conteúdo especifique esta intenção.

Os *intents* podem ser explícitos, em que determina que o componente a ser executado já é definido explicitamente e o outro tipo é o implícito, sendo o sistema operacional que faz a escolha do componente, que com base em algumas regras, define naquele momento qual componente responderá melhor para aquela intenção.

Em relação ao ciclo de vida, os *broadcasts receivers* contém somente um método de ciclo de vida que é chamado no instante em que chega uma mensagem à ele, sendo o componente considerado ativo somente no momento da execução deste método, ou seja, enquanto reage à mensagem (MARTINS, 2009). Portanto, quando chega ao receptor uma mensagem de broadcast, o *Android* aciona o seu método *onReceive()*, passando o objeto *intent* dentro da mensagem, onde permanece ativo até o retorno do *onReceive()*, quando se torna inativo a partir deste momento (MACK, 2010).

Content Providers:

Este componente é responsável pela persistência dos dados no que diz respeito à gravação e acesso de informações com o uso de uma base de dados, como por exemplo o SQ*Lite*, no qual dispõe estes dados para atividades e aplicativos (MACK, 2010).

Com a função de compartilhar dados entre aplicativos, estes componentes rodam por meio de uma URI, sendo que aplicativos específicos possuem a responsabilidade pelo atendimento de determinados padrões de URI, permitindo que serviços que não se conheçam consigam se comunicar. Apresenta fundamental importância quanto ao aspecto de segurança, pois controlam a restrição no acesso aos dados de programas não autorizados. (UZEJKA, 2011).

Armazenamento no *Android*

Ao contrário dos sistemas operacionais para *desktop*, que normalmente oferecem um sistema de arquivos comum, no *Android* todos os dados são visíveis somente para a aplicação dona, ao passo que, para que outras aplicações possam acessar estas informações, deve-se utilizar um componente do tipo *content provider*, como visto anteriormente.

No *Android*, é possível realizar o armazenamento dos dados de várias maneiras, como por meio de um mecanismo chamado de preferências, em que possibilita o armazenamento dos tipos primitivos, sendo esta opção normalmente usada para armazenar as preferências do usuário. É possível também guardar dados diretamente no aparelho ou em um dispositivo de memória removível, usando arquivos.

Como alternativa, através do *SQLite* (MARTINS, 2009), que consiste em um banco de dados que serve como repositório para o armazenamento e manipulação de informações, por intermédio dos métodos para criar, excluir, executar comandos SQL e executar também algumas outras tarefas rotineiras de banco de dados (MACK, 2010), possibilita o suporte ao acesso de operações de rede, que podem ser usados tanto no armazenamento quanto na requisição de dados.

Neste capitulo foi possível fazer a análise da plataforma que pretende-se usar no modelo proposto deste trabalho, na parte que diz respeito aos dispositivos móveis, o *Android*. Pôde-se entender como ocorreu o surgimento desta plataforma, constatando que tratava-se principalmente de uma necessidade que as empresas líderes no mercado da tecnologia móvel tinham em desenvolver uma plataforma que fosse aberta e única para os dispositivos móveis, oferecendo um produto de qualidade ao usuário final e que também facilitasse a criação de aplicações para os desenvolvedores.

Quanto à arquitetura, pôde-se compreender como que é formada a sua composição, apresentando um modelo baseado nos mesmos princípios da camada OSI, em formato de pilha, na qual foi elucidada a dinâmica básica, desde a camada de mais baixo nível, no *kernel* do Linux, até a última camada, de aplicação, operada pelo usuário final.

Nesta dinâmica foi comentada a importância de todas as camadas da arquitetura no que tange à abstração do hardware, no fornecimento de serviços para o suprimento de recursos do desenvolvimento de aplicações, eficiência na execução de processos quanto ao consumo dos recursos de hardware por parte da máquina virtual, além dos benefícios de reusabilidade de componentes no desenvolvimento, entre outras vantagens.

No que se refere à segurança, foi explicado principalmente que o controle é feito à nível de hardware, onde sua arquitetura provem a segurança, como ARM, *TrustZone*, também à nível de sistema operacional, no qual todos os acessos aos recursos passam pela mediação do núcleo da plataforma, portanto, acessos sujeitos aos controles de segurança

do sistema operacional e à nível de execução de processos, onde existem permissões de acesso, pois todas as aplicações executam em ambiente confinado.

Em relação aos componentes do *Android*, foi verificado quais são os principais utilizados pela plataforma, podendo-se compreender o processo de uma atividade, os eventos que a compõe, a dinâmica desde sua criação até sua destruição, os serviços que desempenham processamentos em segundo plano, os *broadcast receivers*, que atuam como uma espécie de gerenciador de notificações, sendo importante para aplicativos que dependem de certos eventos e os *content providers*, que possuem a responsabilidade com a persistência dos dados na plataforma.

4. INFORMAÇÕES DO PRONTUÁRIO MÉDICO

A definição da maior parte das informações do prontuário médico deste modelo de sistema tiveram como base alguns procedimentos e modelos adotados pelos socorristas, como enfermeiros e bombeiros, por exemplo. Estes procedimentos fazem parte do atendimento pré-hospitalar, no que diz respeito à avaliação do paciente.

Atendimento Pré-Hospitalar

Conforme Oliveira (2001), major do Corpo de Bombeiros Militar de Santa Catarina descreve, entre os métodos de avaliação do paciente, encontra-se o método da avaliação dirigida, que numa das etapas usadas, verifica-se a etapa da entrevista, em que o socorrista conversa com o paciente, procurando obter informações deste, de familiares ou testemunhas, buscando entender o tipo de lesão ou enfermidade, assim como demais dados relevantes.

De acordo com Oliveira (2001), quando o paciente estiver inconsciente, o socorrista deve buscar informações com as pessoas presentes no local, devendo conduzir os questionamentos de forma ordenada, entre eles:

1) O nome da vítima
2) Se a vítima tem alguma doença ou problema de saúde?
3) Se alguém sabe se a vítima toma algum remédio ou é alérgica?

Para a situação que o paciente está consciente, deve o socorrista se utilizar das seguintes perguntas-chave e dirigi-las a ele:

1) Nome e idade (se for menor, deve contatar os pais ou adulto conhecido)
2) O que aconteceu? (para constatar a natureza da lesão ou doença)
3) Você tem algum problema de saúde?
4) Você tem tomado algum remédio?
5) Você é alérgico a alguma coisa?

Oliveira (2001) relata ainda que mais recentemente está sendo adotado um modelo de entrevista simplificado, chamado SAMPLE, sendo cada letra da palavra a representação de uma pergunta ao paciente, ou seja:

Sinais e sintomas (O que está errado?)

Alergias (Você é alérgico a algum tipo de substância ou alimento?)

Medicações (Você toma algum tipo de remédio?)

Passado médico (Você está realizando algum tratamento médico?)

Líquidos e alimentos (Você ingeriu alguma coisa recentemente?)

Eventos relacionados com o trauma ou doença (O que aconteceu?)

Na fase de entrevista, Couto (s.d.), engenheiro agrônomo da Universidade Federal Rural do Rio de Janeiro, referindo-se especialmente ao riscos de acidentes na zona rural, complementa destacando que as perguntas nesta etapa podem estar relacionadas a:

- Causas e hora do acidente
- Conhecimento ou parentesco da vítima
- Indicação de antídotos e endereços úteis
- Idade, hábitos, doenças e remédios usados pelo paciente
- Etc.

Diante da análise dos procedimentos realizados no atendimento pré-hospitalar, definiu-se, no Quadro 4.1, como ficaria a definição das informações do prontuário médico de um indivíduo:

Quadro 4.1 Definição dos dados do prontuário médico

Dados de Identificação	**Dados Vitais**	**Dados Complementares**
UID da *tag*	Número do SUS	Planos de Saúde
Nome	Tipo Sanguíneo	Hospitais
CPF	Alergias	Históricos Hospitalares
Idade	Doenças	Hábitos
	Medicamentos	Médicos
	Contatos	Endereços Úteis
	Ult. Atualização	Observações

Fonte: do autor

5. REQUISITOS E MODELAGEM DO SISTEMA

5.1 Requisitos Externos

5.1.1 Premissas do sistema

Hipoteticamente falando, o modelo de sistema proposto poderia ser patrocinado pelo governo, no qual representaria uma base de informações médicas básicas da população. Para tanto, o governo necessitaria mobilizar esforços no sentido de implementar este sistema com uma estrutura capaz de suportar grandes demandas de informação, tanto no sentido de inclusões, atualizações e consultas a esta grande base de dados.

Entre essas demandas estariam a efetivação do cadastro das entidades e/ou profissionais autorizados a usar o sistema, assim como manter atualizadas as tabelas genéricas do sistema, como as tabelas de alergias, doenças, medicamentos, planos de saúde, hospitais e hábitos, na tentativa de se utilizar de termos e nomes padrões, a fim de minimizar erros e inconsistências no processo de cadastro dos dados dos indivíduos.

Também seria importante coordenar esforços para a aquisição e distribuição de aparelhagem (dispositivos móveis, leitores RFID, as *tags*, aplicadores das *tags*, entre outros) para as entidades e profissionais de saúde (postos, hospitais, ambulâncias, os próprios agentes), além de capacitar e treinar estes profissionais para o uso do sistema e para os procedimentos de aplicação/remoção do *chip* encapsulado nas pessoas.

5.1.2 A inclusão do cadastro da pessoa

Antes de efetuar o cadastro

Antes de dirigir-se a um posto de saúde ou hospital para realizar seu cadastro e subsequentemente a incisão subcutânea da *tag*, a pessoa deve ser instruída a estar munida de todas (ou ao menos as mais relevantes) informações sobre a sua saúde, atendendo as seguintes informações:

Informações vitais e de identificação:

-Identidade (será utilizado o nome, CPF e a data de nascimento, para a idade)

-Número do SUS

-Tipo Sanguíneo

-Alergias

-Doenças

-Medicamentos

-Contatos

Informações complementares

-Plano de Saúde

-Hospital

-Histórico hospitalar

-Nome do(s) médico(s)

-Hábitos

-Endereços Úteis

O cadastro

Com toda a documentação em mãos, o agente de saúde pode começar a realizar o cadastro da pessoa. Acessando um portal na web com acesso a base de dados desse sistema, o agente, através de sua autenticação (*login*), entrará com todas as informações citadas acima, além da informação da *tag* que será implantada no indivíduo (que poderá ser o código único da *tag* ou o CPF da pessoa, que neste caso, seria gravado criptografado tanto na *tag*, quanto no banco, onde no banco conteria também a chave), observações e a data da última atualização (neste caso, a data de cadastro).

Uma vez realizado o cadastro, a pessoa está inclusa no sistema, podendo ter seus principais dados médicos disponíveis para os agentes de saúde realizar a consulta (por meio do portal ou pela leitura de seu *chip*) e para possíveis atualizações (somente pelo portal).

Aplicação subcutânea da *tag*

Após efetuar o cadastro da pessoa, esta passaria depois pela intervenção cirúrgica para incisão da *tag* de biovidro (que deve possuir propriedades antimigratórias), em locais previamente estabelecidos e definidos por padrão para esta aplicação. Este

último quesito é importante pelo fato do leitor a ser utilizado para a leitura da *tag* operar à curtas distâncias (de um a três cm), o que tornaria o trabalho do agente de saúde mais difícil no processo de localização da *tag*, o que seria contraprodutivo diante de uma situação emergencial.

Um destes locais para a incisão da *tag* poderia ser entre os dedos polegar e o indicador, por exemplo, mas outros pontos (por volta de mais um ou dois) poderiam ser definidos, para um caso de alguma impossibilidade de tal pessoa receber a aplicação da *tag* em somente um local determinado, oferecendo dessa forma uma alternativa.

A aplicação da *tag* poderia ser realizada em postos de saúde, sendo o procedimento executado por um agente do posto, instruído sobre o método de aplicação da *tag*, além de garantir a integridade física do indivíduo, desta forma inibindo uma possível infecção, por exemplo.

5.1.3 A atualização do cadastro da pessoa

Como o estado de saúde de uma pessoa é um processo dinâmico, é necessário que seja possível realizar atualizações no cadastro desta, como por exemplo, se este indivíduo descobriu que é alérgico a um determinado medicamento ou que possa ter uma nova informação de seu histórico hospitalar, entre outras situações.

Para esta situação, a pessoa deve procurar um posto de saúde ou hospital (munida com a documentação pertinente as novas informações) para que o agente de saúde autorizado realize a atualização de seu cadastro pelo portal do sistema.

Substituição da *tag*

A ideia para esse sistema é que a *tag* seja inserida na pessoa uma única vez, permanecendo nela por toda a sua vida, mas por algum motivo pode ocorrer da *tag* apresentar alguma falha (o que deve ser raro) ou que a mesma possa ter sido danificada (por alguma lesão no local do *chip* que porventura o afetou também).

Neste caso, a pessoa se encaminharia a um posto ou hospital e se submeteria a uma intervenção cirúrgica pelo profissional habilitado para remover a *tag* danificada e colocar uma nova. Para o sistema, bastaria atualizar o código da *tag* para o valor da nova *tag*.

5.1.4 A identificação da pessoa

Uma vez que a pessoa realizou seu cadastro, juntamente com o implante da cápsula, passa a estar apta para ser identificada por um leitor RFID. Entretanto, a identificação também poderá ser manual, uma vez que a pessoa também poderá ser identificada informando seu CPF.

A identificação por meio do CPF seria para minimizar a necessidade de leitores RFID nos locais que fazem o acesso pelo portal (nos postos e hospitais), partindo-se do pressuposto que a pessoa estará munida de sua identidade. Para as consultas via dispositivo móvel, também será possível realizar a consulta de seus dados médicos via CPF, uma vez que representa uma alternativa no caso de não ser possível realizar a leitura da *tag* por motivo de qualquer natureza.

Para a leitura por RF, o processo consiste na aproximação de um a três cm do leitor nos locais pré-estabelecidos, na qual dentro de instantes a informação da *tag* deve aparecer no campo correspondente do aplicativo do dispositivo móvel ou do portal da web, desde que o foco esteja sobre o devido campo, possibilitando a identificação.

Uma vez que o indivíduo foi identificado, o sistema do dispositivo móvel do agente retornará os dados médicos que haviam sido inseridos no cadastro deste indivíduo, em ordem de relevância, (informações de identificação, informações vitais e informações complementares).

5.1.5 A exclusão do cadastro da pessoa

Remoção da *tag*

Quanto à remoção da *tag*, por algum motivo, pode acontecer da pessoa desejar não ter mais a cápsula inserida no seu corpo. Para isso, precisaria dirigir-se a um posto ou hospital e solicitar pela sua remoção, podendo o indivíduo solicitar em seguida a exclusão do seu cadastro, ficando de fora do sistema. Uma outra situação que poderia ocorrer é da pessoa desejar remover a *tag* mas continuar com seu cadastro ativo, possibilitando a consulta de seus dados médicos por meio de seu CPF.

5.2 Casos de Uso

5.2.1 Processo de Cadastro

Para que se possa ter uma compreensão mais ampla de como será a dinâmica do sistema, foi utilizado o recurso de casos de uso da UML (tradução, Linguagem de Modelagem Unificada), tornando-se esta linguagem, nos últimos anos, a linguagem padrão de modelagem utilizada internacionalmente pela indústria de engenharia de software (GUEDES, 2011). Portanto, foram elaborados dois diagramas para possibilitar uma visão macro do sistema. Para a construção dos diagramas foi utilizado o software livre *StarUML*.

O primeiro diagrama corresponde aos processos de cadastro, a entrada e atualização das informações para possibilitar o objetivo principal que são as consultas aos dados do indivíduo. De forma bem sucinta, o diagrama funciona da seguinte forma:

1. O agente de saúde (ator) chama o caso de uso Efetuar Login para entrar no sistema e possibilitar a execução das principais operações de banco de dados.
2. Uma vez logado, o agente poderá chamar os casos de uso Inserir Cadastro, Atualizar Cadastro, Excluir Cadastro ou Buscar Cadastro.
3. Se chamar Inserir Cadastro, precisará obrigatoriamente (<<*include*>>) chamar os casos de uso Inserir Dados Identificadores e Inserir Dados Vitais. Poderá chamar o caso de uso Inserir Dados Complementares (<<*extend*>>). Precisará (<<*include*>>) chamar os casos de uso Validar CPF que inclui Criptografar CPF. Poderá (<<*extend*>>) chamar o caso de uso Buscar Cadastro, para verificar se já existe antes de inserir um novo.
4. Se chamar Atualizar Cadastro, precisará (<<*include*>>) chamar o caso de uso Buscar Cadastro, para escolher qual cadastro sofrerá a atualização. Poderá (<<*extend*>>) chamar os dados de uso Inserir Dados Identificadores, Inserir Dados Vitais, Inserir Dados Complementares ou Validar CPF que inclui Criptografar CPF, para o caso de ser realizada a edição do CPF da pessoa.
5. Se chamar Excluir Cadastro, precisará (<<*include*>>) chamar o caso de uso Buscar CPF, para escolher qual cadastro sofrerá a exclusão.
6. Se chamar Buscar Cadastro, se localizar um cadastro, poderá chamar os casos de uso Atualizar Cadastro ou Excluir Cadastro, se não localizar, poderá chamar o caso de uso Inserir Cadastro.

Abaixo, na Figura 5.1, pode-se observar as etapas que envolvem o processo de cadastro:

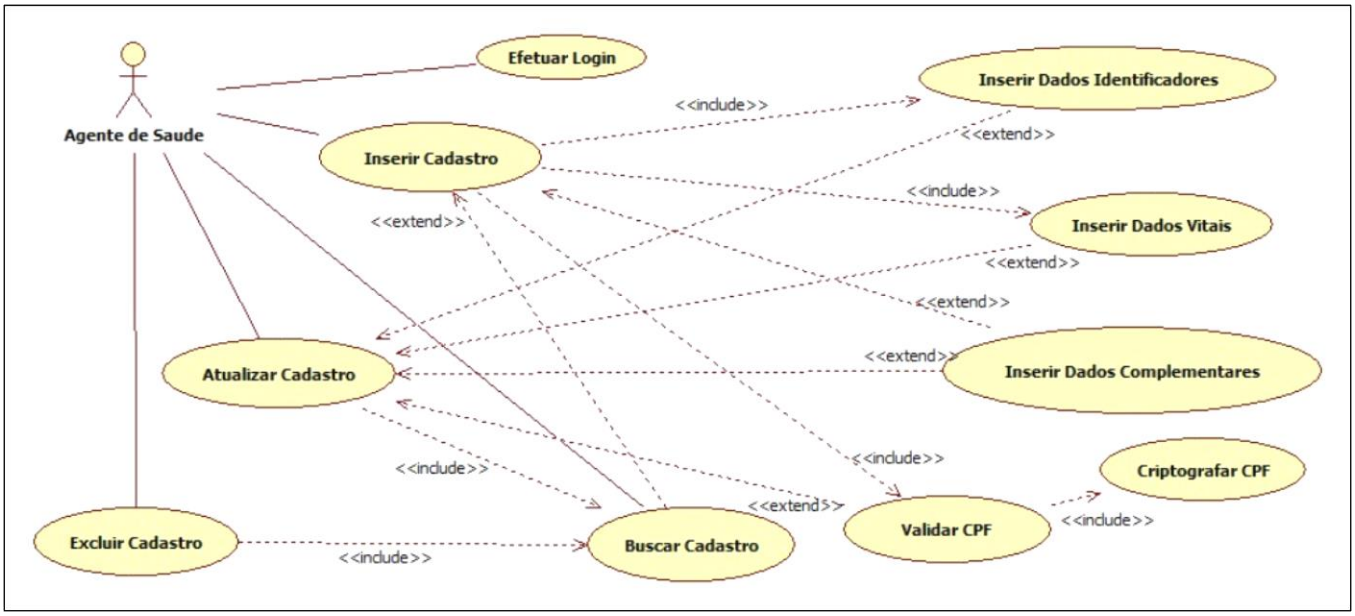


Figura 5.1: Diagrama de Casos de Uso – Cadastro
Fonte: do autor

Contudo, o diagrama de casos de uso consegue fornecer uma visão mais resumida da dinâmica de um sistema, mas cada caso de uso possui um detalhamento de como será sua execução, descrevendo seu resumo, o seu fluxo principal, alternativo ou de exceção, pré-condições, entre outros detalhamentos. Abaixo, pode-se verificar os quadros 5.1 à 5.10 que compõe cada um dos casos de uso deste diagrama.

Quadro 5.1: Caso de Uso – Efetuar Login

Nome do Caso de Uso	**Efetuar Login**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente se autenticar ao sistema
Pré-Condições	1. Deve estar conectado à internet
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Informar o usuário e a senha	
	2. Verificar se o nome de usuário e senha estão corretos
	3. Entrar no sistema
Restrições/Validações	1. O agente precisa estar cadastrado no sistema
	2. O usuário e senha precisam estar corretos
Fluxo de Exceção – *Login* inválido	
Ações do Ator	Ações do Sistema
	1. Informar ao agente que o usuário e/ou senha estão incorreto(s)

Fonte: do autor

Quadro 5.2: Caso de Uso – Inserir Cadastro

Nome do Caso de Uso	**Inserir Cadastro**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente realizar a criação do cadastro
Pré-Condições	O agente deve estar logado no sistema
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Iniciar um cadastro novo	
2. Executar caso de uso Inserir Dados Identificadores	
3. Executar caso de uso Inserir Dados Vitais	
4. Iniciar salvamento do cadastro	
	5. Executar caso de uso Validar CPF
Restrições/Validações	
Fluxo Alternativo I – Inserir Dados Complementares	
Ações do Ator	Ações do Sistema
1. Executar caso de uso Inserir Dados Complementares	
Fluxo Alternativo II – Buscar Cadastro	
Ações do Ator	Ações do Sistema
1. Executar caso de uso Buscar Cadastro	

Fonte: do autor

Quadro 5.3: Caso de Uso – Excluir Cadastro

Nome do Caso de Uso	**Excluir Cadastro**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente realizar a exclusão do cadastro
Pré-Condições	O agente deve estar logado no sistema
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Executar caso de uso Buscar Cadastro	
	2. Perguntar ao agente se realmente deseja excluir o cadastro
3. Excluir o cadastro	
Restrições/Validações	

Fonte: do autor

Quadro 5.4: Caso de Uso – Atualizar Cadastro

Nome do Caso de Uso	**Atualizar Cadastro**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente realizar a atualização do cadastro
Pré-Condições	O agente deve estar logado no sistema
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Executar caso de uso Buscar Cadastro	
2. Realizar a edição do cadastro	
3. Salvar o cadastro	
Restrições/Validações	
Fluxo Alternativo I – Inserir Dados Identificadores	
Ações do Ator	Ações do Sistema
1. Executar caso de uso Inserir Dados Identificadores	
2. Salvar o cadastro	
Fluxo Alternativo II – Inserir Dados Vitais	
Ações do Ator	Ações do Sistema
1. Executar caso de uso Inserir Dados Vitais	
2. Salvar o cadastro	
Fluxo Alternativo III – Inserir Dados Complementares	
Ações do Ator	Ações do Sistema
1. Executar caso de uso Inserir Dados Complementares	
2. Salvar o cadastro	
Fluxo Alternativo IV – Validar CPF	
Ações do Ator	Ações do Sistema
1. Editar o CPF	
2. Iniciar o salvamento do cadastro	
	3. Executar caso de uso Validar CPF

Fonte: do autor

Quadro 5.5: Caso de Uso – Inserir Dados Identificadores

Nome do Caso de Uso	**Inserir Dados Identificadores**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente informar os dados de identificação da pessoa
Pré-Condições	1. O agente deve estar criando ou editando o cadastro
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Informar os dados de identificação da pessoa, sendo eles: Código do *chip*, Nome, CPF e a idade	
Restrições/Validações	1. O preenchimento dos campos são obrigatórios
Fluxo de Exceção – Campos não preenchidos	
Ações do Ator	Ações do Sistema
	1. Informar ao agente que existem campos que dever ser preenchidos

Fonte: do autor

Quadro 5.6: Caso de Uso – Inserir Dados Vitais

Nome do Caso de Uso	**Inserir Dados Vitais**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente informar os dados vitais da pessoa
Pré-Condições	1. O agente deve estar criando ou editando o cadastro
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Informar os dados vitais da pessoa, sendo eles: Número do SUS, Tipo Sanguíneo, Alergias, Doenças, Medicamentos e Contatos	
Restrições/Validações	1. O preenchimento dos campos Número do SUS e Tipo Sanguíneo são obrigatórios
Fluxo de Exceção – Campos não preenchidos	
Ações do Ator	Ações do Sistema
	1. Informar ao agente que existem campos (Número do SUS e Tipo Sanguíneo) que dever ser preenchidos

Fonte: do autor

Quadro 5.7: Caso de Uso – Inserir Dados Complementares

Nome do Caso de Uso	**Inserir Dados Complementares**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente informar os dados complementares da pessoa
Pré-Condições	1. O agente deve estar criando ou editando o cadastro
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Informar os dados complementares da pessoa, sendo eles: Plano(s) de Saúde, Hospital(is), Histórico(s) Hospitalar(es), Hábitos, Nome do(s) Médico(s)	
Restrições/Validações	

Fonte: do autor

Quadro 5.8: Caso de Uso – Buscar Cadastro

Nome do Caso de Uso	**Buscar Cadastro**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para buscar um cadastro
Pré-Condições	1. O agente deve estar logado no sistema
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Informar o CPF da pessoa	
	2. Retornar o cadastro da pessoa
Restrições/Validações	1. O cadastro deve ser localizado
Fluxo de Exceção – Cadastro inexistente	
Ações do Ator	Ações do Sistema
	1. Informar ao agente que o cadastro não foi localizado

Fonte: do autor

Quadro 5.9 Caso de Uso – Validar CPF

Nome do Caso de Uso	**Validar CPF**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para validar um CPF informado
Pré-Condições	1. O agente deve estar salvando um cadastro
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
	1. Aplicar algoritmo de validação do CPF
	2. Executar caso de uso Criptografar CPF
Restrições/Validações	1. O CPF informado deve ser válido
Fluxo de Exceção – CPF inválido	
Ações do Ator	Ações do Sistema
	1. Informar ao agente que o CPF é inválido

Fonte: do autor

Quadro 5.10 Caso de Uso – Criptografar CPF

Nome do Caso de Uso	**Criptografar CPF**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para criptografar um CPF validado
Pré-Condições	1. O sistema deve ter validado o CPF
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
	1. Aplicar algoritmo de criptografia no CPF, aplicando uma chave de 128 bits baseada no próprio CPF
	2. Finalizar o salvamento do cadastro
Restrições/Validações	

Fonte: do autor

5.2.2 Processo de Consulta

Uma vez determinado o comportamento da alimentação do sistema, passa-se para o segundo momento e o mais útil em termos práticos para os agentes de saúde, que são as consultas aos dados médicos das pessoas. Para possibilitar uma visão geral dos processos que envolvem a consulta, foi elaborado o segundo diagrama, que procura demonstrar como ocorrerão as interações do agente com o sistema.

Assim como os passos do diagrama de cadastro, segue-se abaixo de forma resumida quais as principais etapas deste diagrama:

1. O agente de saúde (ator) chama o caso de uso Efetuar Login para entrar no sistema e possibilitar a consulta dos dados de uma pessoa
2. O agente poderá chamar o caso de uso Ler Chip RFID, que estende (<<*extend*>>) o caso de uso Buscar Registro
3. Da mesma forma, o agente poderá chamar o caso de uso Informar CPF, que também estende (<<*extend*>>) o caso de uso Buscar Registro
4. Se chamar o caso de uso Buscar Registro, com base em um dos casos de uso que o estende (Ler Chip RFID e Informar CPF), deverá chamar o caso de uso Acessar WS (<<*include*>>) para realizar a busca do registro da pessoa consultada.
5. Após chamar o caso de uso Buscar Registro, passará a chamar o caso de uso Receber Retorno, que precisa (<<*include*>>) chamar o caso de uso Acessar WS, para obter o resultado de sua busca.
6. O caso de uso Acessar WS aciona o WS (ator secundário), que poderá chamar os casos de uso Buscar Por ID do Chip, Buscar Por CPF ou Buscar por CPF Criptog.
7. Se chamar o caso de uso Buscar Por ID do Chip, chamará em seguida o caso de uso Enviar Retorno por ser uma alternativa deste (<<*extend*>>), onde o caso de uso Enviar Retorno precisará chamar (<<*include*>>) o caso de uso Acessar WS para que seja enviado o retorno ao agente de saúde.
8. Se chamar o caso de uso Buscar por CPF, realizará os mesmos eventos do passo acima (7), a partir da chamada do caso de uso Enviar Retorno.
9. Se chamar o caso de uso Buscar Por CPF Criptog., este precisará chamar (<<*include*>>) o caso de uso Validar CPF Criptog., sendo que após, passa a realizar as mesmas etapas do passo 7, a partir da chamada do caso de uso Enviar Retorno.

Abaixo, na Figura 5.2, pode-se observar os passos que envolvem os processos de consulta de dados de um indivíduo:

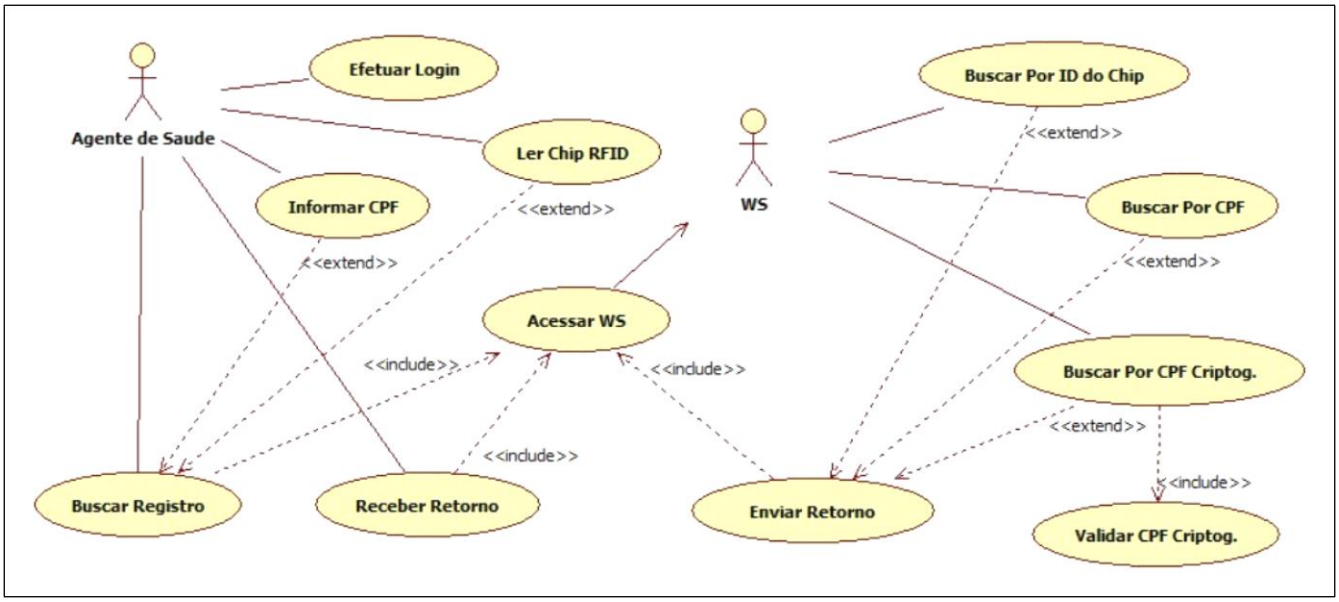


Figura 5.2: Diagrama de Casos de Uso – Consulta
Fonte: do autor

Entretanto, da mesma forma como para o diagrama de cadastros, este diagrama possui uma descrição mais detalhada dos casos de uso que o constituem, podendo-se observar abaixo a relação dos quadros 5.11 à 5.20 que traz estas informações:

Quadro 5.11: Caso de Uso – Efetuar Login

Nome do Caso de Uso	**Efetuar Login**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente se autenticar ao sistema
Pré-Condições	1. Deve estar conectado à internet
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Informar o usuário e a senha	
	2. Verificar se o nome de usuário e senha estão corretos
	3. Entrar no sistema
Restrições/Validações	1. O agente precisa estar cadastrado no sistema
	2. O usuário e senha precisam estar corretos
Fluxo de Exceção – *Login* inválido	
Ações do Ator	Ações do Sistema
	1. Informar ao agente que o usuário e/ou senha estão incorreto(s)

Fonte: do autor

Quadro 5.12: Caso de Uso – Ler Chip RFID

Nome do Caso de Uso	**Ler Chip RFID**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	
Resumo	Descreve os passos necessários para o agente realizar a leitura do *chip* RFID
Pré-Condições	1. O agente deve estar logado no sistema
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Posicionar o leitor RFID à curta distância dos locais pré-determinados	
	2. Capturar o código do *chip* através do leitor RFID
	3. Apresentar no campo de busca do registro o código do *chip* RFID
Restrições/Validações	

Fonte: do autor

Quadro 5.13: Caso de Uso – Buscar Registro

Nome do Caso de Uso	**Buscar Registro**
Caso de Uso Geral	
Ator Principal	Agente de Saúde
Atores Secundários	WS (*Webservice*)
Resumo	Descreve os passos necessários para o agente buscar o registro
Pré-Condições	1. O agente deve estar logado no sistema
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Enviar os dados de identificação para o WS	
	2. Executar caso de uso Acessar WS
Restrições/Validações	O campo de busca do registro não pode estar vazio
Fluxo de Exceção – Campo não pode estar vazio	
Ações do Ator	Ações do Sistema
	1. O sistema deve informar que o campo deve possuir ao menos 10 caracteres

Fonte: do autor

Quadro 5.14: Caso de Uso – Acessar WS

Nome do Caso de Uso	**Acessar WS**
Caso de Uso Geral	
Ator Principal	WS (*Webservice*)
Atores Secundários	Agente de Saúde
Resumo	Descreve os passos necessários para o agente trocar informações com o WS
Pré-Condições	1. O agente deve ter executado o caso de uso Buscar Registro
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Executar caso de uso Buscar por ID do *chip*	
2. Executar caso de uso Buscar por CPF	
3. Executar caso de uso Buscar por CPF Criptog.	
	4. Executar caso de uso Enviar Retorno
	5. Executar caso de uso Receber Retorno
Restrições/Validações	O WS deve retornar uma resposta ao agente
Fluxo de Exceção – Demora no retorno do WS	
Ações do Ator	Ações do Sistema
	1. Caso exceda o tempo de espera, o aplicativo deve informar que não foi possível obter uma resposta

Fonte: do autor

Quadro 5.15: Caso de Uso – Buscar Por ID do Chip

Nome do Caso de Uso	**Buscar Por ID do Chip**
Caso de Uso Geral	
Ator Principal	WS (*Webservice*)
Atores Secundários	Agente de Saúde
Resumo	Descreve os passos necessários para o WS buscar o registro pelo ID do *chip*
Pré-Condições	
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Verificar se o valor passado pelo WS possui 10 dígitos	
	2. Pesquisar no banco de dados pelo ID do *chip*
	3. Retornar o registro ou mensagem que indique que não foi localizado
	4. Executar caso de uso Enviar Retorno
Restrições/Validações	O valor passado pelo WS deve conter 10 dígitos
Fluxo de Exceção – Formato de valor diferente do ID do *chip*	
Ações do Ator	Ações do Sistema
	1. Executar caso de uso Acessar WS para realizar busca por CPF normal ou CPF criptografado

Fonte: do autor

Quadro 5.16: Caso de Uso – Buscar Por CPF

Nome do Caso de Uso	**Buscar Por CPF**
Caso de Uso Geral	
Ator Principal	WS (*Webservice*)
Atores Secundários	Agente de Saúde
Resumo	Descreve os passos necessários para o WS buscar o registro pelo CPF da pessoa
Pré-Condições	
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Verificar se o valor passado pelo WS possui 11 dígitos	
	2. Pesquisar no banco de dados pelo CPF da pessoa
	3. Retornar o registro ou mensagem que indique que não foi localizado
	4. Executar caso de uso Enviar Retorno
Restrições/Validações	O valor passado pelo WS deve conter 11 dígitos
Fluxo de Exceção – Formato de valor diferente do CPF	
Ações do Ator	Ações do Sistema
	1. Executar caso de uso Acessar WS para realizar busca por ID do *chip* ou CPF criptografado

Fonte: do autor

Quadro 5.17: Caso de Uso – Buscar Por CPF Criptog.

Nome do Caso de Uso	**Buscar Por CPF Criptog.**
Caso de Uso Geral	
Ator Principal	WS (*Webservice*)
Atores Secundários	Agente de Saúde
Resumo	Descreve os passos necessários para o WS buscar o registro pelo CPF criptografado da pessoa
Pré-Condições	
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. Verificar se o valor passado pelo WS é superior à 11 dígitos	
	2. Pesquisar no banco de dados pelo CPF criptografado da pessoa
	3. Executar caso de uso Validar CPF Criptog.
	4. Retornar o registro ou mensagem que indique que não foi localizado
	5. Executar caso de uso Enviar Retorno
Restrições/Validações	O valor passado pelo WS deve ser superior à 11 dígitos
Fluxo de Exceção – Formato de valor diferente do CPF criptografado	
Ações do Ator	Ações do Sistema
	1. Executar caso de uso Acessar WS para realizar busca por ID do *chip* ou CPF normal

Fonte: do autor

Quadro 5.18: Caso de Uso – Validar CPF Criptog.

Nome do Caso de Uso	**Validar CPF Criptog.**
Caso de Uso Geral	
Ator Principal	WS (*Webservice*)
Atores Secundários	Agente de Saúde
Resumo	Descreve os passos necessários para que o CPF criptografado da pessoa seja validado
Pré-Condições	O caso de uso Buscar Por CPF Criptog. deve ter encontrado o registro
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
	1. Descriptografar o CPF com a chave contida no registro
	2. Verificar se o CPF descriptografado é igual ao CPF normal contido no registro
	3. Continuar a execução do caso de uso Buscar por CPF Criptog.
Restrições/Validações	O CPF descriptografado deve ser igual ao CPF normal contido no registro
Fluxo de Exceção – CPF descriptografado não confere com CPF normal do registro	
Ações do Ator	Ações do Sistema
	1. Retornar mensagem indicando que um registro foi localizado mas que falhou na sua validação
	2. Executar caso de uso Enviar Retorno

Fonte: do autor

Quadro 5.19: Caso de Uso – Enviar Retorno

Nome do Caso de Uso	**Enviar Retorno**
Caso de Uso Geral	
Ator Principal	WS (*Webservice*)
Atores Secundários	Agente de Saúde
Resumo	Descreve os passos necessários para o que o WS receba um retorno de sua busca
Pré-Condições	
Pós-Condições	
Fluxo Alternativo I – Retorno obtido pela busca por ID do *chip*	
Ações do Ator	Ações do Sistema
	1. O retorno é obtido pelo caso de uso Busca por ID do *chip*
	2. Encaminha o retorno para o caso de uso Acessar WS
Fluxo Alternativo II – Retorno obtido pela busca por CPF normal	
Ações do Ator	Ações do Sistema
	1. O retorno é obtido pelo caso de uso Busca por CPF
	2. Encaminha o retorno para o caso de uso Acessar WS
Fluxo Alternativo III – Retorno obtido pela busca por CPF criptografado	
Ações do Ator	Ações do Sistema
	1. O retorno é obtido pelo caso de uso Busca por CPF Criptog.
	2. Encaminha o retorno para o caso de uso Acessar WS

Fonte: do autor

Quadro 5.20 Caso de Uso – Receber Retorno

Nome do Caso de Uso	**Receber Retorno**
Caso de Uso Geral	
Ator Principal	WS (*Webservice*)
Atores Secundários	Agente de Saúde
Resumo	Descreve os passos necessários para que o WS entregue o resultado da busca ao agente
Pré-Condições	
Pós-Condições	
Fluxo Principal	
Ações do Ator	Ações do Sistema
	1. Após obter o resultado do caso de uso Enviar Retorno, o caso de uso Acessar WS encaminha o retorno ao aplicativo do agente
	2. O agente recebe o registro da sua busca
Restrições/Validações	O agente deve receber o registro de sua busca
Fluxo de Exceção I – O registro não foi localizado	
Ações do Ator	Ações do Sistema
	1. Informar ao agente que não foi possível encontrar o registro procurado
Fluxo de Exceção II – O retorno apresentou um erro	
Ações do Ator	Ações do Sistema
	1. Informar ao agente que houve um erro na busca do registro, mostrando a mensagem do erro ocorrido

Fonte: do autor

5.3 Diagrama de Classes

Através do diagrama de classes é possível que se possa ter uma visão estática do sistema, demonstrando como que as classes interagem entre si, os relacionamentos que as compõe, apresentando os atributos e métodos que integram as classes do sistema (GUEDES, 2011).

O sistema para o cadastro e consulta de dados médicos foi modelado, chegando-se ao número de 13 classes responsáveis pelo armazenamento dos dados por meio de seus atributos e das operações a serem realizadas através de seus métodos. Abaixo, serão descritas as classes que constituem o sistema, juntamente com seus atributos e métodos.

***Agente*:** Esta classe é responsável por armazenar os principais dados dos agentes de saúde, sendo composta pelos atributos *registro*, que armazena seu registro perante o órgão que regulariza os profissionais da saúde, *nome*, como o próprio termo diz, refere-se ao nome do agente, *login*, representando seu nome para o sistema e *senha*, para o seu *login* no sistema. Todos os atributos são do tipo *String* (texto).

Esta classe possui dois métodos, o primeiro é *efetuarLogin(String, String): int*, possuindo a função de realizar o *login* do agente junto ao sistema, passando por parâmetros seu nome de usuário e senha e tendo como retorno um inteiro, sendo que zero representa que sua autenticação falhou no sistema e um que conseguiu autenticar-se com sucesso.

O outro método está definido como *buscarPaciente(String): String*, possuindo a função de realizar a busca no sistema de um determinado paciente, tendo como parâmetro a identificação da pessoa e recebendo como retorno um inteiro, no qual zero representa que não foi localizado e um, que foi localizado.

Esta classe associa-se com outras duas classes, sendo a primeira classe *Paciente*, sendo que, quando o agente acessar o sistema para realização de cadastros, pode registrar um ou mais pacientes, representando a multiplicidade de 1..*(um para muitos). A segunda classe é a *WebService*, que apresenta a associação quando o agente está realizando consultas dos dados médicos de um determinado paciente via um dispositivo móvel, podendo o agente acessar o *webservice*, caracterizando a multiplicidade de 1..1(um para um).

***WebService*:** Esta classe tem o objetivo de retornar a consulta de todos os dados médicos de um indivíduo identificado, não possuindo atributos, somente métodos.

Esta classe possui três métodos, sendo eles *buscarDadosIdentif(String): String*, que tem a finalidade de retornar os dados identificadores do prontuário da pessoa, possuindo como parâmetro a identificação do indivíduo e como retorno os dados de identificação desta.

O outro método chama-se *buscarDadosVitais(String): String*, que possui o propósito de retornar os dados importantes do paciente, ou seja, como o próprio nome diz, os dados vitais. Assim como o método anterior, possui o parâmetro da identificação da pessoa e como retorno os dados vitais dela.

O último método denomina-se *buscarDadosComplem(String): String*, no qual tem o objetivo de retornar os dados complementares do indivíduo, tratando-se de informações de importância secundária para o agente que realiza o atendimento. Também possui como parâmetro a identificação da pessoa, sendo o seu retorno os dados complementares do paciente.

Assim como a classe *Agente*, esta classe associa-se a outras duas classes, sendo elas a classe *Agente*, podendo ser acessada por um ou mais agentes, caracterizando a multiplicidade 1..*. É por meio desta associação que o agente pode chamar os métodos desta classe, passando a identificação da pessoa consultada.

A segunda classe é a *Paciente*, podendo-se consultar nesta classe um ou mais pacientes, apresentando a multiplicidade 1..*. Com esta associação é realizada a chamada aos métodos da *Paciente* e por conseguinte a chamada aos métodos das demais classes ligadas a esta última, representando ao final o retorno dos métodos da classe *WebService*.

***Paciente*:** Pode-se dizer que esta classe é a mais importante do sistema, pois guarda os dados principais do paciente e possui conexão com todas as classes do sistema. Ela possui os atributos *uid_tag*, que armazena o código de identificação do *chip*, para o caso do *chip* ser do tipo somente leitura, depois aparecem os atributos *nome* e *cpf* do paciente. Após vem o atributo *chave_cpf*, que representa a chave de criptografia do CPF e *cpf_criptog*, que armazena o CPF criptografado. Todos estes atributos citados são do tipo *String* (texto).

Após vem o atributo *idade*, do tipo inteiro, *num_sus*, do tipo *String* (texto) onde guarda o número do SUS do paciente e *tipo_sanguineo*, também do tipo *String* (texto). Depois aparece o atributo *ult_atualizacao*, do tipo *Date* (data) com o propósito de manter o registro da data da última atualização que foi realizada no cadastro do indivíduo e por fim, o atributo *observacoes*, do tipo *String* (texto), com a finalidade de armazenar qualquer dado que não se enquadre nos atributos existentes.

Esta classe possui cinco métodos, sendo eles *registrarPaciente(): int*, que possui o propósito de realizar o cadastro do indivíduo, possuindo um retorno do tipo inteiro, sendo o valor zero a representação da falha no registro e o valor um representando o sucesso no cadastro, tendo a sua chamada feita pela classe *Agente*. Outro método é o *consultarDadosPac(String): String* com a finalidade de retornar os dados desta classe, contidos em seus atributos, representando o seu retorno e possuindo como parâmetro a identificação do paciente, recebendo a sua chamada pela classe *WebService*.

Os próximos métodos são referentes ao CPF do paciente, sendo eles *validar_cpf(String): int*, com o objetivo de realizar a sua validação, recebendo ele como parâmetro e retornando um inteiro, sendo o valor zero como CPF inválido e o valor um para o caso de ser válido. Sequencialmente vem o método *criptog_cpf(String): String*

com o intuito de realizar a criptografia do CPF, que o recebe como parâmetro e o retorna, mas criptografado. Por fim, aparece o método *descriptog_cpf(String): String*, que possui a função inversa do último método explicado, ou seja, tem o propósito de descriptografar o CPF, contendo como parâmetro o CPF criptografado e o seu retorno descriptografado.

Como citado anteriormente, esta classe possui associação com todas as classes do sistema, entre elas está a associação com a *Agente*, sendo um paciente registrado por um agente, caracterizando a multiplicidade 1..1. Há também a associação com a *WebService*, sendo que um paciente é consultado pelo *webservice*, representando também a multiplicidade 1..1.

Esta classe associa-se com diversas classes que complementam os seus dados, sendo elas a classe *Habitos*, podendo o paciente possuir nenhum ou vários hábitos (0..*), *Medicamentos*, que pode possuir nenhum ou vários medicamentos (0..*), *EnderecosUteis*, possuindo nenhum ou vários endereços úteis (0..*), *Contatos*, podendo ter nenhum ou vários contatos (0..*), *Alergias*, que pode ter nenhuma ou diversas alergias (0..*), *HistoricoHospitalar*, possuindo nenhum ou vários históricos hospitalares (0..*), *Hospitais*, podendo ter nenhum ou vários hospitais, *PlanosSaude*, possuindo nenhum ou vários planos de saúde, *Medicos*, que pode ter nenhum ou diversos médicos e por fim *Doencas*, podendo o paciente possuir nenhuma ou várias doenças.

***Hábitos*:** Esta classe tem a finalidade de armazenar os hábitos de um determinado paciente. Possui como atributo *habito*, do tipo *String* (texto), referente ao armazenamento dos hábitos. Tem como métodos *buscarHabitos(): String*, com o intuito de acessar uma fonte de dados que disponibilizaria através de seu retorno, as opções para o cadastro dos hábitos específicos ao indivíduo, podendo vir de um *webservice* externo, por exemplo.

Depois vem o método *registrarHabitosPac(): int*, que possui o objetivo de realizar o cadastro dos hábitos inerentes à pessoa, retornando o valor zero num caso de erro no cadastro e o valor um sinalizando a conclusão do cadastro. Por último, surge o método *consultarHabitosPac(String): String*, com parâmetro *String* (texto), representando a identificação do paciente e retorno *String* (texto) que objetiva por meio deste último apresentar os hábitos de tal pessoa, sendo acessado pela classe *Paciente*, sendo que um hábito pode pertencer a um ou mais pacientes (1..*).

***Medicamentos*:** Esta classe tem a finalidade de guardar os medicamentos da pessoa cadastrada. Tem como atributo *medicamento*, do tipo *String* (texto), referente ao armazenamento dos medicamentos. Possui os métodos *buscarMedicamentos(): String*, com o propósito de acessar uma fonte de dados que ofereça através de seu retorno, as opções para o cadastro dos medicamentos do paciente.

Após tem o método *registrarMedicamPac(): int*, com a finalidade de efetivar o cadastro dos medicamentos do indivíduo, devolvendo o valor zero num caso de falha do cadastro e o valor um representando a conclusão do cadastro. Por fim, aparece o método *consultarMedicamPac(String): String*, com parâmetro *String* (texto), caracterizando a identificação do paciente e retorno *String* (texto) que visa através do seu retorno apresentar os medicamentos do paciente, sendo ligado pela classe *Paciente*, sendo que um medicamento pode pertencer a um ou mais pacientes (1..*).

***EnderecosUteis*:** Nesta classe fica armazenado os endereços úteis de uma pessoa. Tem como atributos a *descrição*, do tipo *String* (texto), descrevendo do que se trata o endereço. Os demais atributos *logradouro String* (texto), *numero* (inteiro), *complemento, bairro, cidade, uf*, estes últimos do tipo *String* (texto) e *cep* (inteiro) fazem o armazenamento dos dados do endereço informado e por último *telefone* do tipo *String* (texto), guardando o telefone do endereço.

Possui o método *registrarEndUtilPac(): int*, com a função de realizar o cadastro dos endereços úteis do paciente, retornando o valor zero para falha do cadastro e o valor um para o sucesso do cadastro. Também contém o método *consultarEndUteisPac(String): String*, com parâmetro *String* (texto) e retorno *String* (texto) que tem a função de retornar os endereços úteis do indivíduo, sendo associado pela classe *Paciente*, no qual o endereço útil pertence a um paciente (1..1).

***Contatos*:** Na classe *Contatos* tem-se o objetivo de armazenar os contatos do indivíduo. Possui os atributos *nome*, *parentesco* e *telefone*, todos do tipo *String* (texto), com a função de guardar os principais dados do contato da pessoa.

Possui o método *registrarContatoPac(): int*, com a função de cadastrar os contatos da pessoa, tendo como retorno o valor zero para sinalizar falha do cadastro e o valor um para o cadastro concluído com êxito. Também contém o método *consultarContatoPac(String): String*, com parâmetro *String* (texto), tratando-se da identificação do paciente e retorno *String* (texto) que tem a função de retornar os contatos

da pessoa, estando conectado com a classe *Paciente*, sendo que o contato pertence a um paciente (1..1).

***Alergias*:** Esta classe possui a função de armazenar as alergias do indivíduo cadastrado. Tem como atributo *descricao*, do tipo *String* (texto), referente à descrição da alergia. Contém os métodos *buscarAlergias(): String*, com o objetivo de buscar de uma fonte de dados os tipos de alergias existentes por intermédio do seu retorno, apresentando as opções para o cadastro das alergias do paciente.

Em seguida existe o método *registrarAlergiaPac(): int*, com o intuito de realizar o cadastro das alergias da pessoa, retornando o valor zero para o erro de cadastro e o valor um para a finalização do cadastro. Por último, tem o método *consultarAlergiasPac(String): String*, com parâmetro *String* (texto), sendo a identificação do paciente e retorno *String* (texto) que busca retornar as alergias da pessoa, estando conectada na classe *Paciente*, pois uma alergia pode pertencer a um ou mais pacientes (1..*).

***HistoricoHospitalar*:** Esta classe é do tipo associativa, sendo necessária para as situações onde existem atributos relacionados à associação que não podem ser guardados por nenhuma das classes envolvidas (GUEDES, 2011), sendo elas *Hospitais*, que pode possuir diversos históricos hospitalares e *Paciente*, que pode possuir vários hospitais.

Essa classe visa guardar o histórico hospitalar da pessoa. Apresenta o atributo *descricao*, do tipo *String* (texto), com a função de reter uma descrição do histórico hospitalar.

Possui o método *registrarHistHospPac(): int*, que possui o propósito de registrar os históricos hospitalares do indivíduo, contendo como retorno o valor zero para indicar falha do cadastro e o valor um para indicar o sucesso do cadastro. Possui ainda o método *consultarHistHospPac(String): String*, com parâmetro *String* (texto), tratando-se da identificação do paciente e como retorno o tipo *String* (texto) que objetiva retornar os históricos hospitalares do paciente consultado.

Possui associação entre as classes *Paciente*, cujo histórico hospitalar pertence a um paciente (1..1) e a classe *Hospitais*, na qual o histórico hospitalar pertence a um hospital (1..1).

***Hospitais*:** Aqui nesta classe encontram-se guardados os hospitais do paciente. Tem como atributo *descricao*, do tipo *String* (texto), referente ao hospital que o paciente

tem convênio ou que possa ter algum histórico. A classe associativa *HistoricoHospitais* só possuirá registro se derivar do registro desta classe.

Contém os métodos *buscarHospitais(): String*, que fica à cargo de acessar uma fonte de dados e de retornar os hospitais existentes, exibindo as opções para o cadastro dos hospitais do paciente.

Em seguida localiza-se o método *registrarHospPac(): int*, com o intuito de realizar o cadastro dos hospitais da pessoa, possuindo como retorno o valor zero para erro no cadastro e o valor um para efetivação do cadastro.

Por último, contempla o método *consultarHospPac(String): String*, com parâmetro *String* (texto), referenciando a identificação do paciente e retorno *String* (texto) que tem a função de retornar os hospitais registrados para o indivíduo, existindo relação com a classe *Paciente*, sendo que um hospital pode pertencer a um ou mais pacientes (1..*).

***PlanosSaude*:** Esta classe caracteriza-se por armazenar os planos de saúde do paciente em questão. Tem como atributos *plano* e *telefone*, ambos do tipo *String* (texto), referentes ao plano que o paciente tem e o telefone deste convênio. Contém os métodos *buscarPlanos(): String*, que possui a tarefa de acessar uma fonte de dados e retornar os tipos de planos existentes, oferecendo as opções para o cadastro dos planos do indivíduo.

Após encontra-se o método *registrarPlanoPac(): int*, com a finalidade de executar o cadastro dos planos da pessoa, tendo como retorno o valor zero para a falha no cadastro e o valor um para conclusão do cadastro.

Por fim, tem o método *consultarPlanosPac(String): String*, com parâmetro *String* (texto), representando a identificação do paciente e retorno *String* (texto) que tem como meta retornar os planos cadastrados para a pessoa, possuindo associação com a classe *Paciente*, no qual um plano pode pertencer a um ou mais pacientes (1..*).

***Medicos*:** Na classe *Medicos* tem-se o objetivo de guardar os médicos da pessoa. Possui os atributos *medico*, *especialidade* e *telefone*, todos do tipo *String* (texto), possuindo o papel de armazenar os principais dados do médico do paciente.

Possui o método *registrarMedicoPac(): int*, com propósito de registrar os médicos da pessoa, contendo como retorno o valor zero para indicar erro no cadastro e o valor um simbolizando o cadastro concluído com êxito. Também contém o método

consultarMedicosPac(String): String, com parâmetro *String* (texto), indicando a identificação do paciente e retorno *String* (texto) que tem o objetivo de retornar os médicos do indivíduo, encontrando-se vinculado com a classe *Paciente*, cujo médico pertence a um paciente (1..1).

***Doencas*:** Nesta classe localizam-se armazenado os tipos de doenças que a pessoa possui. Contém o atributos *descrição*, do tipo *String* (texto), representando a descrição da doença que o paciente tem. Incorpora os métodos *buscarDoencas(): String*, que tem o propósito de acessar uma fonte de dados, buscando via seu retorno os tipos de doenças que existem, disponibilizando as opções para o cadastro das doenças do indivíduo.

Subsequentemente localiza-se o método *registrarDoencaPac(): int*, com a função de cadastrar as doenças do indivíduo, cujo seu retorno pode ser o valor zero quando há falha de cadastro ou o valor um para a efetivação do cadastro.

Por fim, aparece o método *consultarDoencasPac(String): String*, com parâmetro *String* (texto), significando a identificação do paciente e retorno *String* (texto) que visa retornar as doenças cadastradas para a pessoa consultada, contendo a associação com a classe *Paciente*, sendo que uma doença pode pertencer a um ou mais pacientes (1..*).

Abaixo, na Figura 5.3, pode-se observar todas as classes, atributos, métodos e associações descritas:

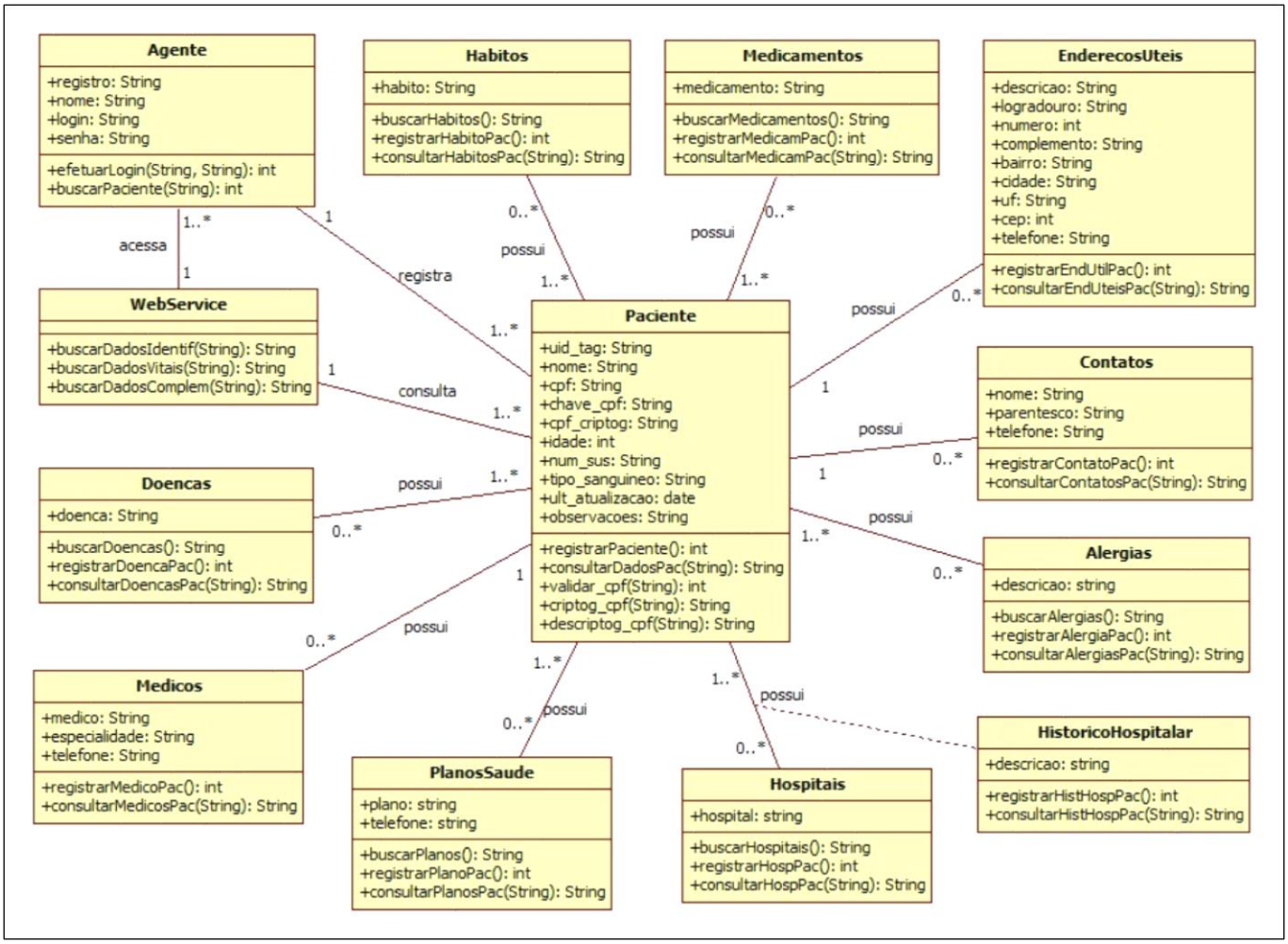


Figura 5.3: Diagrama de Classes
Fonte: do autor

5.4 Diagrama de Objetos

O diagrama de objetos trata-se de um diagrama amplamente vinculado ao diagrama de classes, atuando como um complemento a este. Tem como principal objetivo apresentar uma visão dos valores contidos pelos objetos de um diagrama de classes em um determinado momento da execução do sistema (GUEDES, 2011).

Conforme a Figura 5.4 mostrada abaixo, é possível visualizar a simulação de um cenário de execução do sistema, podendo-se perceber as instâncias de cada classe, juntamente com seus exemplos de valores:

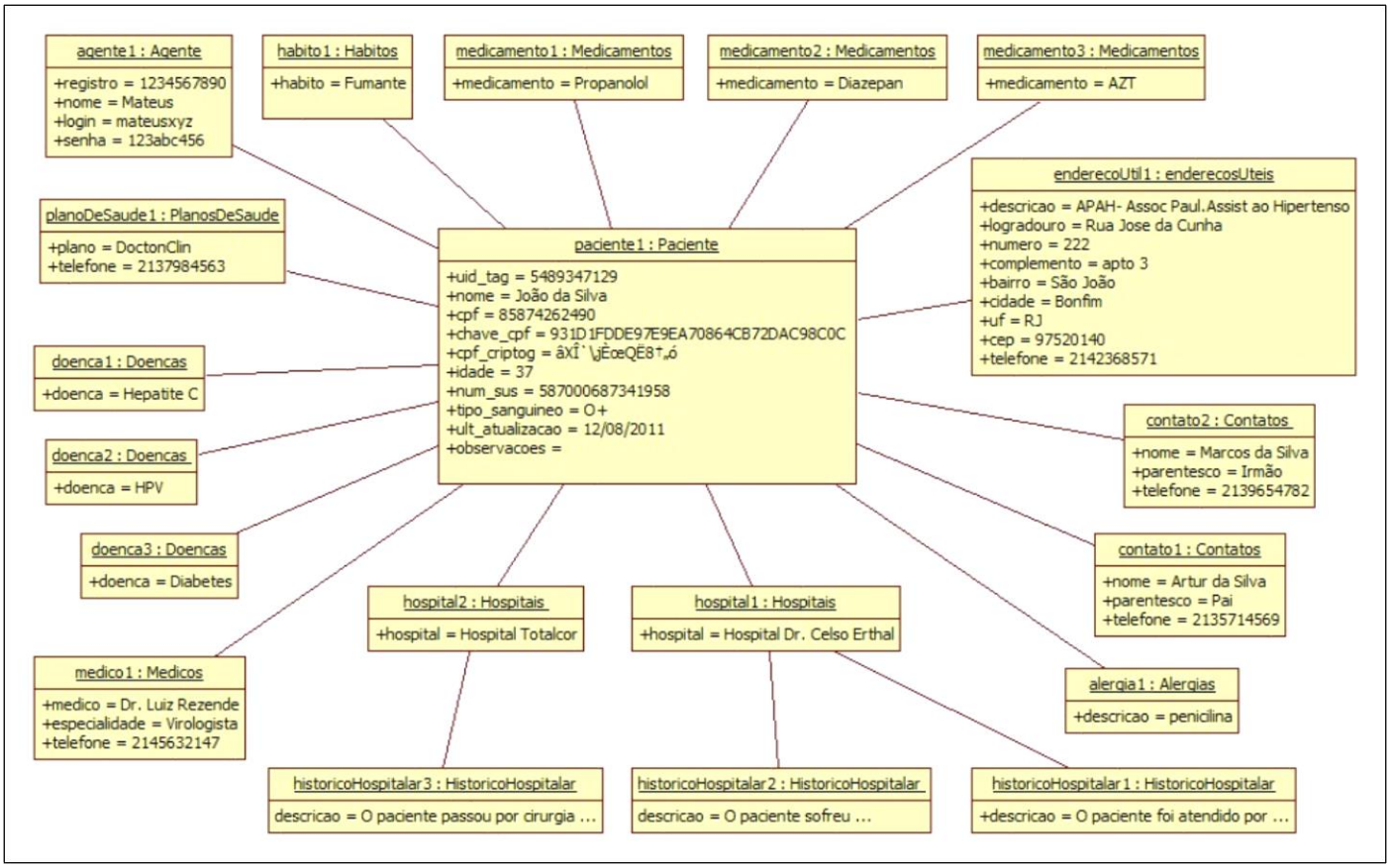


Figura 5.4: Diagrama de Objetos
Fonte: do autor

5.5 Diagrama de Sequência

5.5.1 Cadastro

Este diagrama tem como objetivo estabelecer a sequência de eventos que ocorrem para um determinado processo, indicando quais as mensagens que devem ser enviadas e em qual ordem, estabelecendo a relação entre as mensagens enviadas, métodos a serem chamados, assim como as interações entre os objetos envolvidos (GUEDES, 2011).

O modelo do sistema foi dividido em dois diagramas de sequência, sendo um para o processo do cadastro e outro para a consulta. A seguir será descrito o diagrama de sequência para o cadastro.

Neste diagrama encontra-se o ator, sendo representando pelo agente e envolve todas as classes do sistema, exceto a classe *WebService*, utilizada no diagrama de sequência para consulta. A sequência ficou estabelecida pela seguinte ordem:

1. O agente chama o método *efetuarLogin()* de uma instância da classe *Agente*, realizando a busca de sua autenticação, por cada agente, sinalizada através do fragmento *loop* (tradução Laço), no qual este operador define que um fragmento combinado determina um laço que poderá ser repetido diversas vezes (GUEDES, 2011);
2. O agente obtém o retorno do método *efetuarLogin()*, sinalizando se conseguiu entrar no sistema;
3. Após a autenticação, o agente chama o método *registrarPaciente()* de uma instância da classe *Paciente* onde inicia o cadastro de um paciente;
4. Dentro do objeto da classe *Paciente* é realizada uma autochamada ao método *validar_cpf()*, podendo ocorrer após o momento que o agente informou o CPF da pessoa;
5. Ainda dentro desta instância é realizada outra autochamada ao método *criptog_cpf()*, após ter validado o CPF;
6. O agente realiza a chamada ao método *buscarAlergias()*, cujo objeto da classe *Alergias* deve acessar uma fonte externa, podendo ser um *webservice*.
7. A instância da classe *Alergias* retorna uma lista de alergias, sinalizada através do fragmento *loop*, representando um laço que se repete para cada alergia buscada;
8. O agente envia uma mensagem ao método *registrarAlergiasPac()* de uma instância da classe *Alergias*, para que seja realizado o registro da alergia do paciente, podendo ser efetuado para outras alergias, conforme indica o fragmento *loop*;
9. O agente realiza a chamada ao método *buscarDoencas()*, cujo objeto da classe *Doencas* deve acessar uma fonte externa, podendo ser um *webservice*.
10. A instância da classe *Doencas* retorna uma lista de doenças, sinalizada através do fragmento *loop*, representando um laço que se repete para cada doença buscada;
11. O agente envia uma mensagem ao método *registrarDoencaPac()* de uma instância da classe *Doencas*, para que seja realizado o registro da doença do paciente, podendo ser efetuado para outras doenças, conforme indica o fragmento *loop*;

12. O agente realiza a chamada ao método *buscarMedicamentos()*, cujo objeto da classe *Medicamentos* deve acessar uma fonte externa, podendo ser um *webservice*.
13. A instância da classe *Medicamentos* retorna uma lista de medicamentos, sinalizada através do fragmento *loop*, representando um laço que se repete para cada medicamento buscado;
14. O agente envia uma mensagem ao método *registrarMedicamPac()* de uma instância da classe *Medicamentos*, para que seja realizado o registro do medicamento que o paciente usa, podendo ser efetuado para outros medicamentos, conforme indica o fragmento *loop*;

Abaixo, na Figura 5.5, pode-se observar até este passo como estas sequências estão dispostas no diagrama:

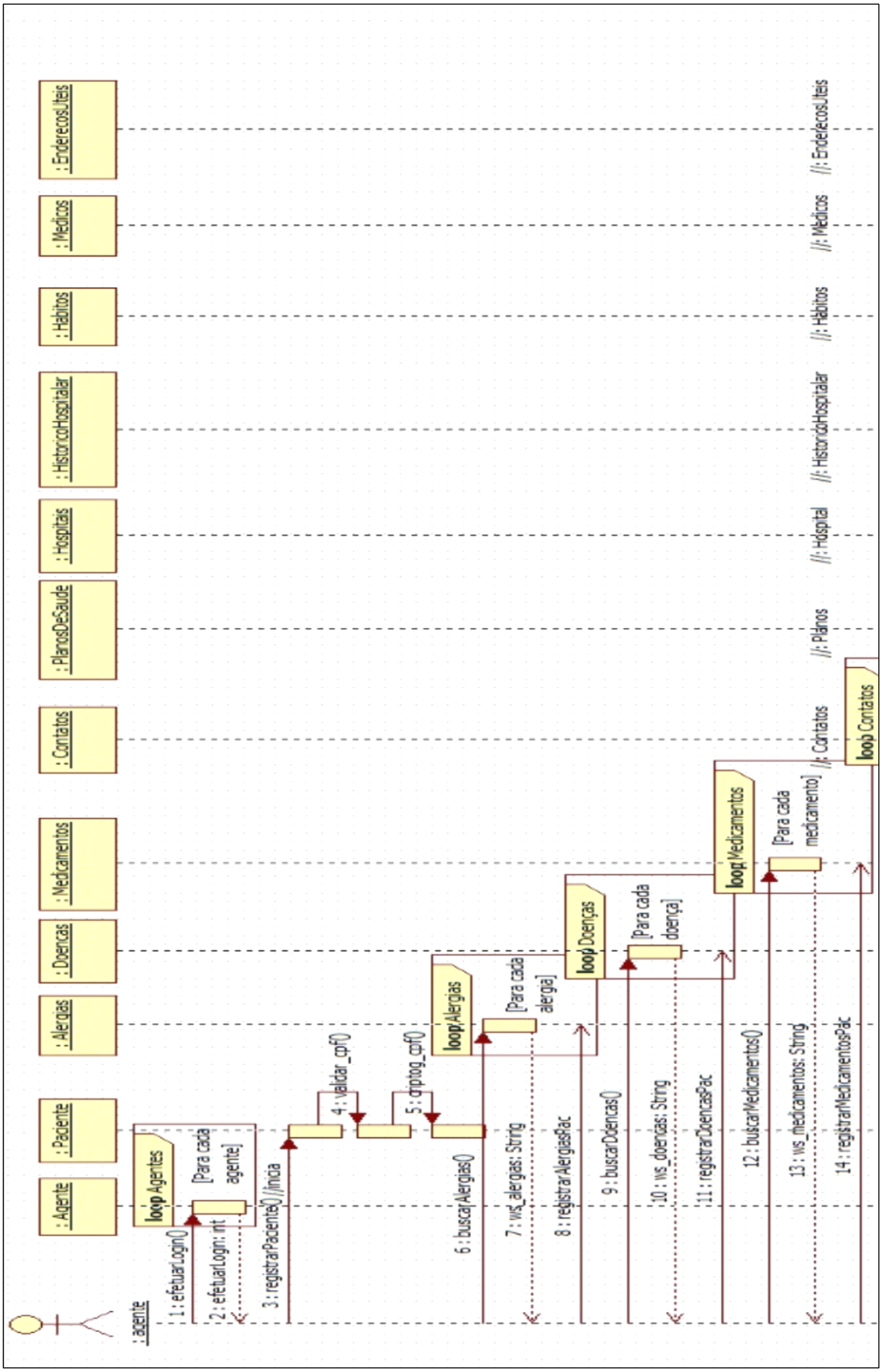


Figura 5.5: Diagrama de Sequência – Cadastro – Parte 1
Fonte: do autor

15. O agente envia uma mensagem ao método *registrarContatoPac()* de uma instância da classe *Contatos*, para que seja realizado o registro do contato do paciente, podendo ser efetuado para outros contatos, conforme indica o fragmento *loop*;
16. O agente realiza a chamada ao método *buscarPlanos()*, cujo objeto da classe *Planos* deve acessar uma fonte externa, podendo ser um *webservice*.
17. A instância da classe *Planos* retorna uma lista de planos, sinalizada através do fragmento *loop*, representando um laço que se repete para cada plano buscado;
18. O agente envia uma mensagem ao método *registrarPlanoPac()* de uma instância da classe *Planos*, para que seja realizado o registro do plano que o paciente é conveniado, podendo ser efetuado para outros planos, conforme indica o fragmento *loop*;
19. O agente realiza a chamada ao método *buscarHospitais()*, cujo objeto da classe *Hospitais* deve acessar uma fonte externa, podendo ser um *webservice*.
20. A instância da classe *Hospitais* retorna uma lista de hospitais, sinalizada através do fragmento *loop*, representando um laço que se repete para cada hospital buscado;
21. O agente envia uma mensagem ao método *registrarHospPac()* de uma instância da classe *Hospitais*, para que seja realizado o registro do hospital que o paciente é cadastrado ou que possua histórico hospitalar, podendo ser efetuado para outros hospitais, conforme indica o fragmento *loop*;
22. O agente envia uma mensagem ao método *registrarHistHospPac()* de uma instância da classe associativa *HistoricoHospitalar*, para que seja realizado o registro do histórico hospitalar do paciente, podendo ser efetuado para outros históricos hospitalares, conforme indica o fragmento *loop*;
23. O agente realiza a chamada ao método *buscarHabitos()*, cujo objeto da classe *Habitos* deve acessar uma fonte externa, podendo ser um *webservice*.
24. A instância da classe *Habitos* retorna uma lista de hábitos, sinalizada através do fragmento *loop*, representando um laço que se repete para cada hábito buscado;
25. O agente envia uma mensagem ao método *registrarHabitosPac()* de uma instância da classe *Habitos*, para que seja realizado o registro do hábito que

o paciente possui, podendo ser efetuado para outros hábitos, conforme indica o fragmento *loop*;

26. O agente envia uma mensagem ao método *registrarMedicosPac()* de uma instância da classe *Medicos*, para que seja realizado o registro do médico do paciente, podendo ser efetuado para outros médicos, conforme indica o fragmento *loop*;
27. O agente envia uma mensagem ao método *registrarEndUtilPac()* de uma instância da classe *EnderecosUteis*, para que seja realizado o registro de um endereço útil do paciente, podendo ser efetuado para outros endereços úteis, conforme indica o fragmento *loop*;
28. Por fim, conforme indicado nesta sequência sugerida, após o registro do endereço útil, representando neste caso o último dado a ser registrado, a instância da classe *Paciente* retorna o status da conclusão do registro do cadastro do paciente;

Abaixo, é possível visualizar na Figura 5.6 a continuidade do diagrama a partir do evento 15. As instâncias das classes que não aparecem, estão sinalizadas como comentário, no formato *//: nome da classe*.

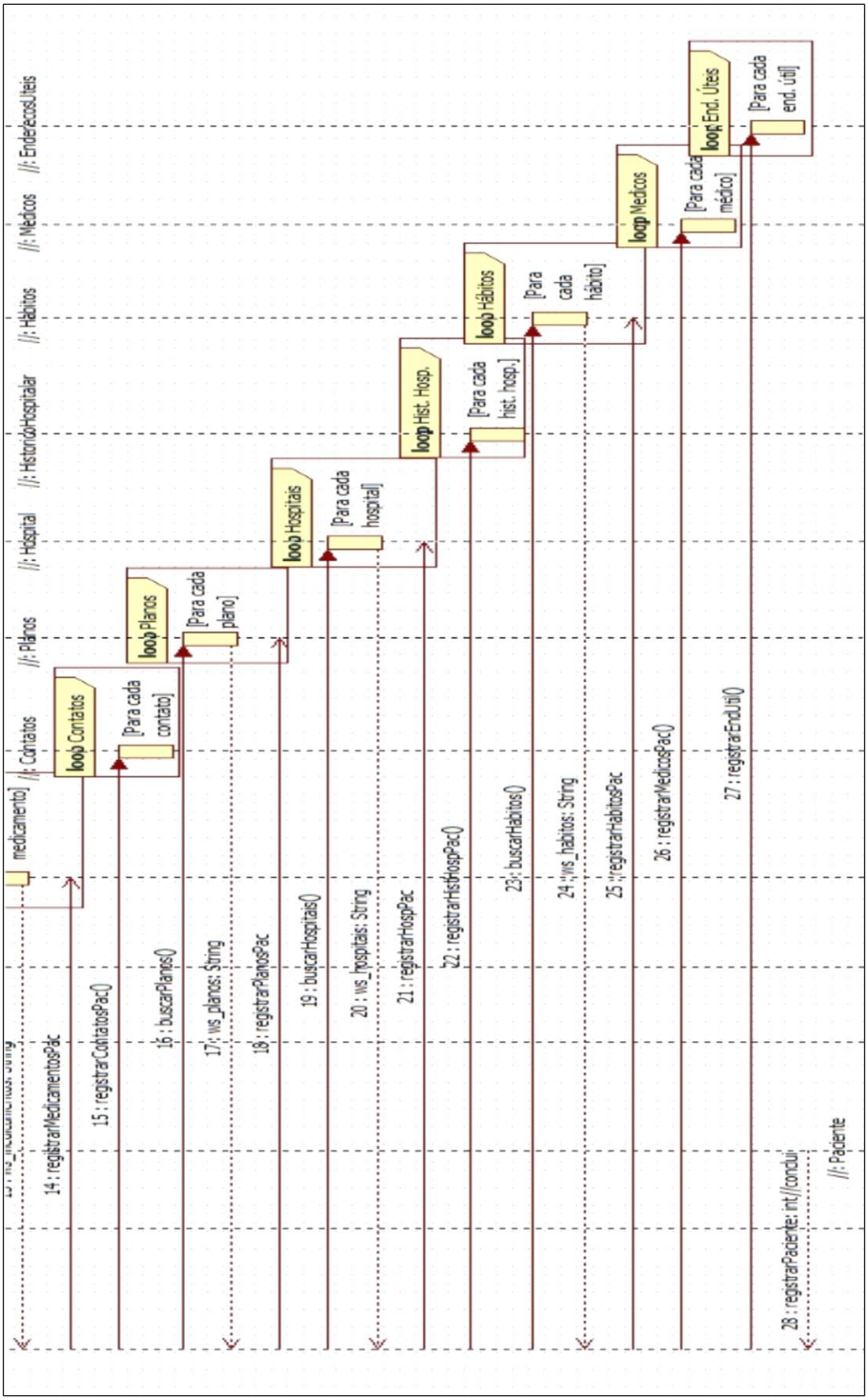


Figura 5.6: Diagrama de Sequência – Cadastro – Parte 2
Fonte: do autor

5.5.2 Consulta

Este representa o segundo diagrama de sequência do modelo do sistema, detalhando os eventos que fazem parte do processo de consulta aos dados do indivíduo. Neste diagrama encontra-se o ator, sendo representando pelo agente e envolve todas as classes do sistema. Em seguida, será descrita a ordem dos eventos deste processo.

1. O agente envia uma mensagem para a chamada do método *efetuarLogin()* de uma instância da classe *Agentes*, passando seu nome de usuário e senha para a classe *WebService* realizar a chamada;
2. A classe *WebService* chama o método *efetuarLogin()*, realizando a busca da autenticação do agente, percorrendo os agentes registrados, sinalizada através do fragmento *loop*;
3. A classe *WebService* obtém o retorno do método *efetuarLogin()* da instância da classe *Agentes*;
4. *WebService* envia o retorno para o agente, sinalizando se conseguiu entrar no sistema;
5. Uma vez autenticado, o agente envia uma mensagem para a chamada do método *buscarPaciente()*, passando a identificação do paciente para a classe *WebService* realizar a chamada;
6. A classe *WebService* chama o método *buscarPaciente()* da instância da classe *Paciente*, realizando a busca da identificação do paciente, percorrendo os pacientes cadastrados, sinalizada através do fragmento *loop*;
7. O próprio objeto da classe *Paciente* recebe o retorno do método *buscarPaciente()*;
8. Esta sequência é opcional, sinalizada pelo fragmento *opt* (tradução Opção), no qual este fragmento determina a escolha de um comportamento, sendo que este comportamento poderá ou não ser executado (GUEDES, 2011). Este passo é realizado para o caso do retorno do método *buscarPaciente()* ter sido realizado com o parâmetro de entrada com CPF criptografado, neste caso, aqui é realizada a autochamada do objeto da classe *Paciente* ao seu método *descriptog_cpf()*;
9. Uma vez localizada a identificação da pessoa, a instância da classe *Paciente* chama o método *buscarDadosIdentif()* da classe *WebService*;

10. A classe *WebService*, através de seu método *buscarDadosIdentif()*, chama o método *consultarDadosPac()* da instância da classe *Paciente*, lembrando que este último método retorna somente os dados contidos nos atributos da classe *Paciente*;
11. A classe *WebService* recebe o retorno do método *consultarDadosPac()* do objeto da classe *Paciente*;
12. O agente recebe o retorno do método *buscarDadosIdentif()* da classe *WebService*;

Na sequência, pode-se visualizar o diagrama fragmentado na Figura 5.7, até o ponto dos eventos aqui descritos:

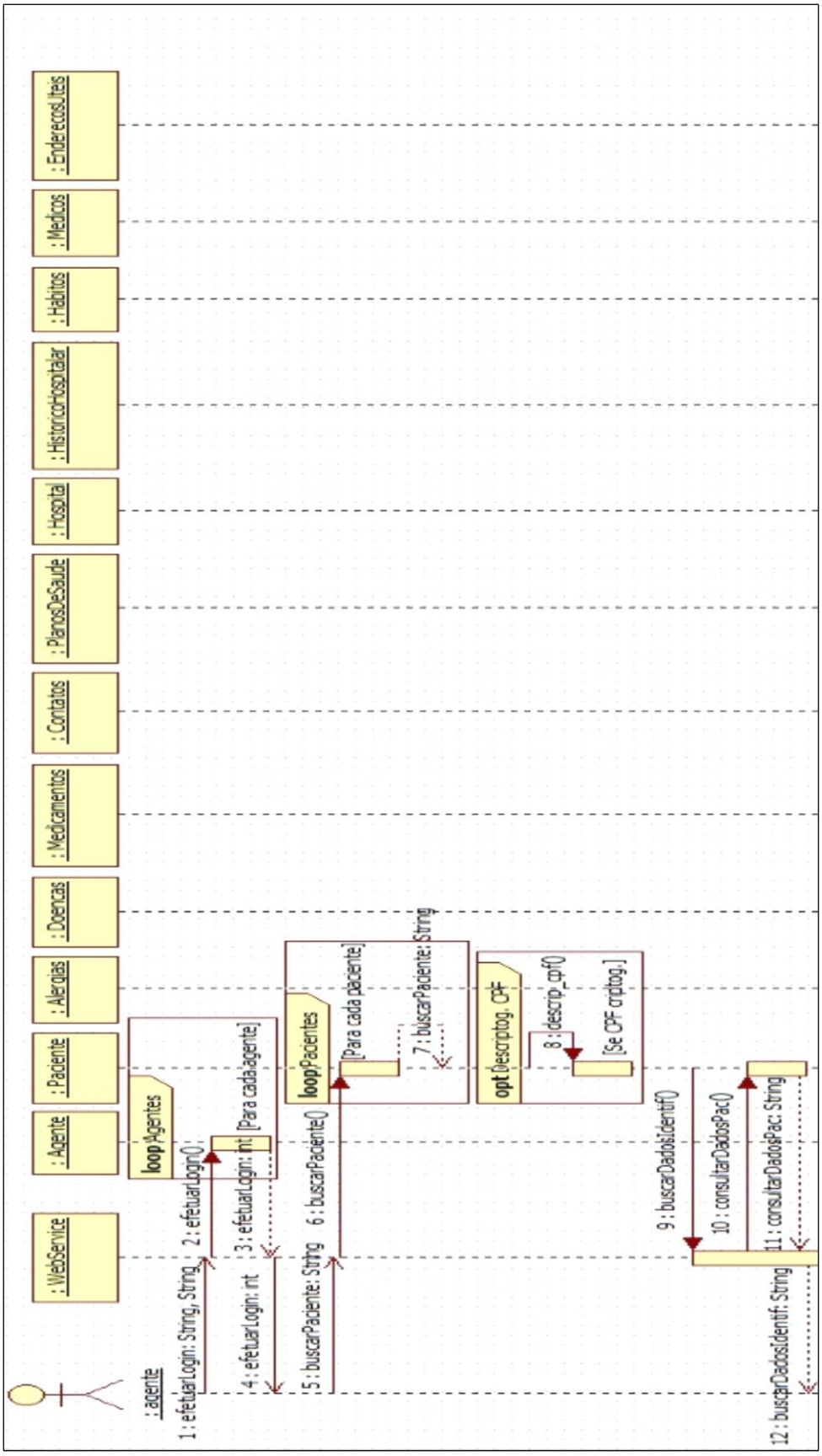


Figura 5.7: Diagrama de Sequência - Consulta – Parte 1
Fonte: do autor

13. Após o retorno do método *buscarDadosIdentif()*, a classe *WebService* realiza uma autochamada ao seu método *buscarDadosVitais()*;
14. A classe *WebService* chama o método *consultarDadosPac()* de uma instância da classe *Paciente*, desta vez para obter os dados pertinentes à categoria de dados vitais do paciente;
15. *WebService* recebe o retorno de *consultarDadosPac()*;
16. A classe *WebService* chama o método *consultarAlergiasPac()* de uma instância da classe *Alergias*, buscando cada registro de alergia do paciente, conforme indica o fragmento *loop*;
17. O método *consultarAlergiasPac()* retorna as alergias do paciente para a classe *WebService*;
18. A classe *WebService* chama o método *consultarDoencasPac()* de uma instância da classe *Doencas*, buscando cada registro de doença do paciente, conforme indica o fragmento *loop*;
19. O método *consultarDoencasPac()* retorna as doenças do paciente para a classe *WebService*;
20. A classe *WebService* chama o método *consultarMedicamPac()* de uma instância da classe *Medicamentos*, buscando cada registro de medicamento do paciente, conforme indica o fragmento *loop*;
21. O método *consultarMedicamPac()* retorna os medicamentos do paciente para a classe *WebService*;
22. A classe *WebService* chama o método *consultarContatosPac()* de uma instância da classe *Contatos*, buscando cada registro de contato do paciente, conforme indica o fragmento *loop*;
23. O método *consultarContatosPac()* retorna os contatos do paciente para a classe *WebService*;
24. O agente recebe o retorno do método *buscarDadosVitais()* da classe *WebService*;

Abaixo, na Figura 5.8, observa-se a continuidade do diagrama fragmentado, englobando as sequências dos passos descritos desde 13 até 24:

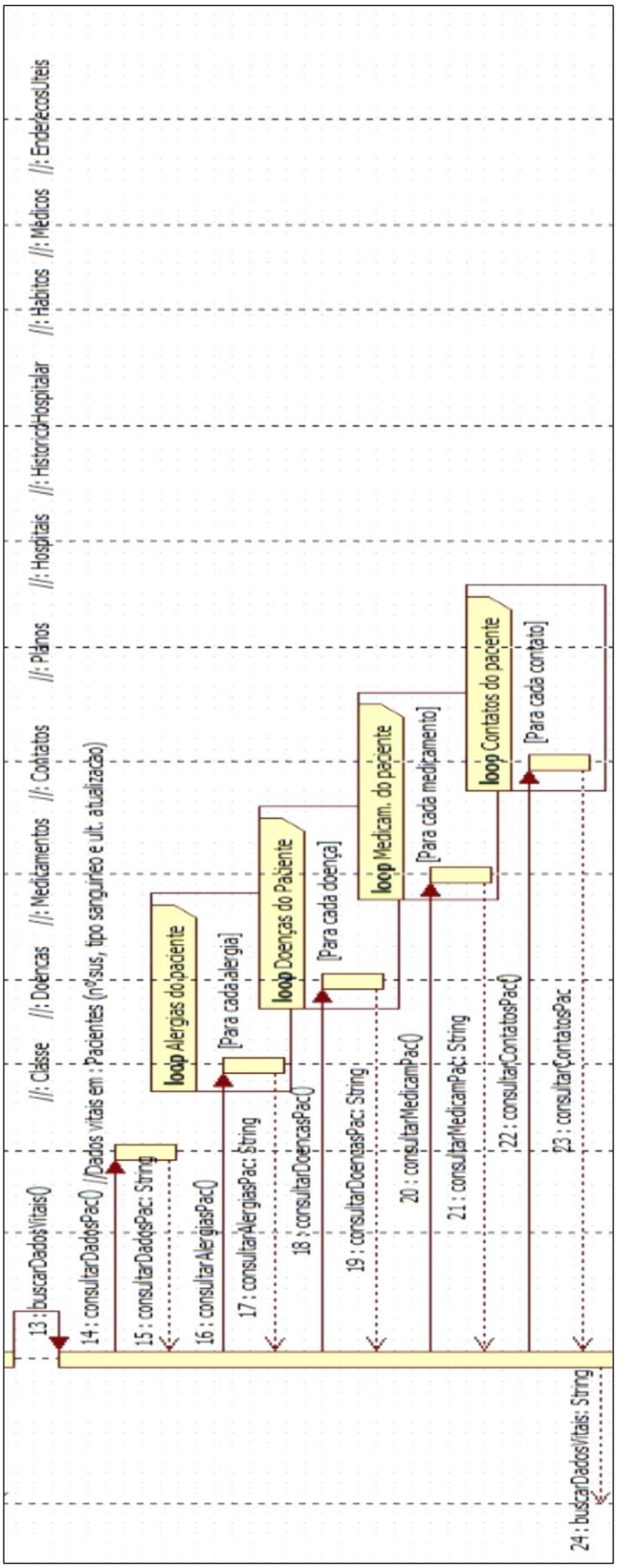


Figura 5.8: Diagrama de Sequência - Consulta – Parte 2
Fonte: do autor

25. Após o retorno do método *buscarDadosVitais()*, a classe *WebService* realiza uma autochamada ao seu método *buscarDadosComplem()*;
26. A classe *WebService* chama o método *consultarPlanosPac()* de uma instância da classe *Planos*, buscando cada registro de plano de saúde do paciente, conforme indica o fragmento *loop*;
27. O método *consultarPlanosPac()* retorna os planos de saúde do paciente para a classe *WebService*;
28. A classe *WebService* chama o método *consultarHospPac()* de uma instância da classe *Hospitais*, buscando cada registro de hospital do paciente, conforme indica o fragmento *loop*;
29. O método *consultarHospPac()* retorna os hospitais do paciente para a classe *WebService*;
30. A classe *WebService* chama o método *consultarHistHospPac()* de uma instância da classe associativa *HistoricoHospitalar*, buscando cada registro de histórico hospitalar do paciente, conforme indica o fragmento *loop*;
31. O método *consultarHistHospPac()* retorna os históricos hospitalares do paciente para a classe *WebService*;
32. A classe *WebService* chama o método *consultarHabitosPac()* de uma instância da classe *Habitos*, buscando cada registro de hábito do paciente, conforme indica o fragmento *loop*;
33. O método *consultarHabitosPac()* retorna os hábitos do paciente para a classe *WebService*;
34. A classe *WebService* chama o método *consultarMedicosPac()* de uma instância da classe *Medicos*, buscando cada registro de médico do paciente, conforme indica o fragmento *loop*;
35. O método *consultarMedicosPac()* retorna os médicos do paciente para a classe *WebService*;
36. A classe *WebService* chama o método *consultarEndUteisPac()* de uma instância da classe *EnderecosUteis*, buscando cada registro de endereço útil do paciente, conforme indica o fragmento *loop*;
37. O método *consultarEndUteisPac()* retorna os endereços úteis do paciente para a classe *WebService*;
38. A classe *WebService* chama o método *consultarDadosPac()* de uma instância da classe *Paciente*, desta vez para obter os dados pertinentes à

categoria de dados complementares do paciente, neste caso, somente as observações;

39. *WebService* recebe o retorno de *consultarDadosPac()*;
40. Por fim, o agente recebe o retorno do método *buscarDadosComplem()* da classe *WebService*;

Na Figura 5.9, abaixo, é possível visualizar no fragmento do diagrama a sequência de passos que encerra a consulta dos dados médicos de um indivíduo:

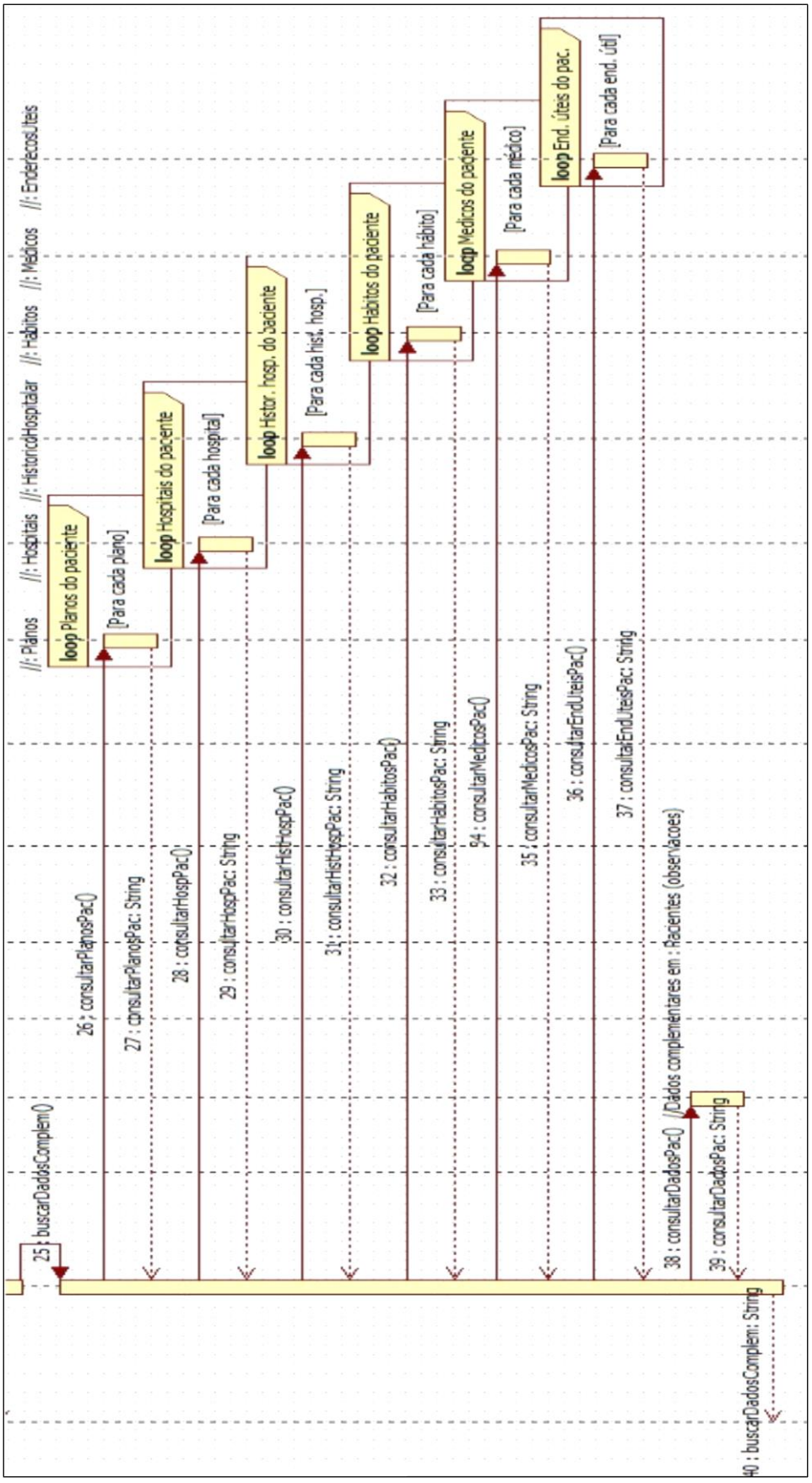


Figura 5.9: Diagrama de Sequência - Consulta – Parte 3
Fonte: do autor

5.6 Diagrama ER

Conforme Heuser (2001) descreve, a abordagem Entidade-Relacionamento (ER) é uma das técnicas de modelagem mais difundidas, sendo que este modelo de dados possui sua representação através de um modelo ER, no qual é representado graficamente por meio de um Diagrama Entidade-Relacionamento (DER). Alguns dos principais conceitos que envolvem a abordagem ER, estão:

Entidade: Tem a finalidade de representar um conjunto de objetos da realidade modelada, sobre os quais serão mantidas as informações no banco de dados;

Relacionamento: Possui o objetivo de apresentar um conjunto de associações entre entidades, possuindo consigo a propriedade de cardinalidade, que possui a função de mostrar o número de ocorrências de um entidade que pode estar associada a uma determinada ocorrência por meio de um relacionamento;

Atributo: É o dado que é vinculado a cada ocorrência de uma entidade ou de um relacionamento (HEUSER, 2001);

Encontram-se, neste modelo ER, dispostas todas as tabelas que compõe o modelo de banco de dados que poderia ser usado para o sistema. Além das tabelas, pode-se observar os campos que as compõem, seus tipos de dados, assim como as chaves utilizadas para manter a integridade de seus registros e realizar os relacionamentos entre as tabelas do sistema. Para a montagem do diagrama ER foi utilizada a ferramenta *DBDesigner 4*.

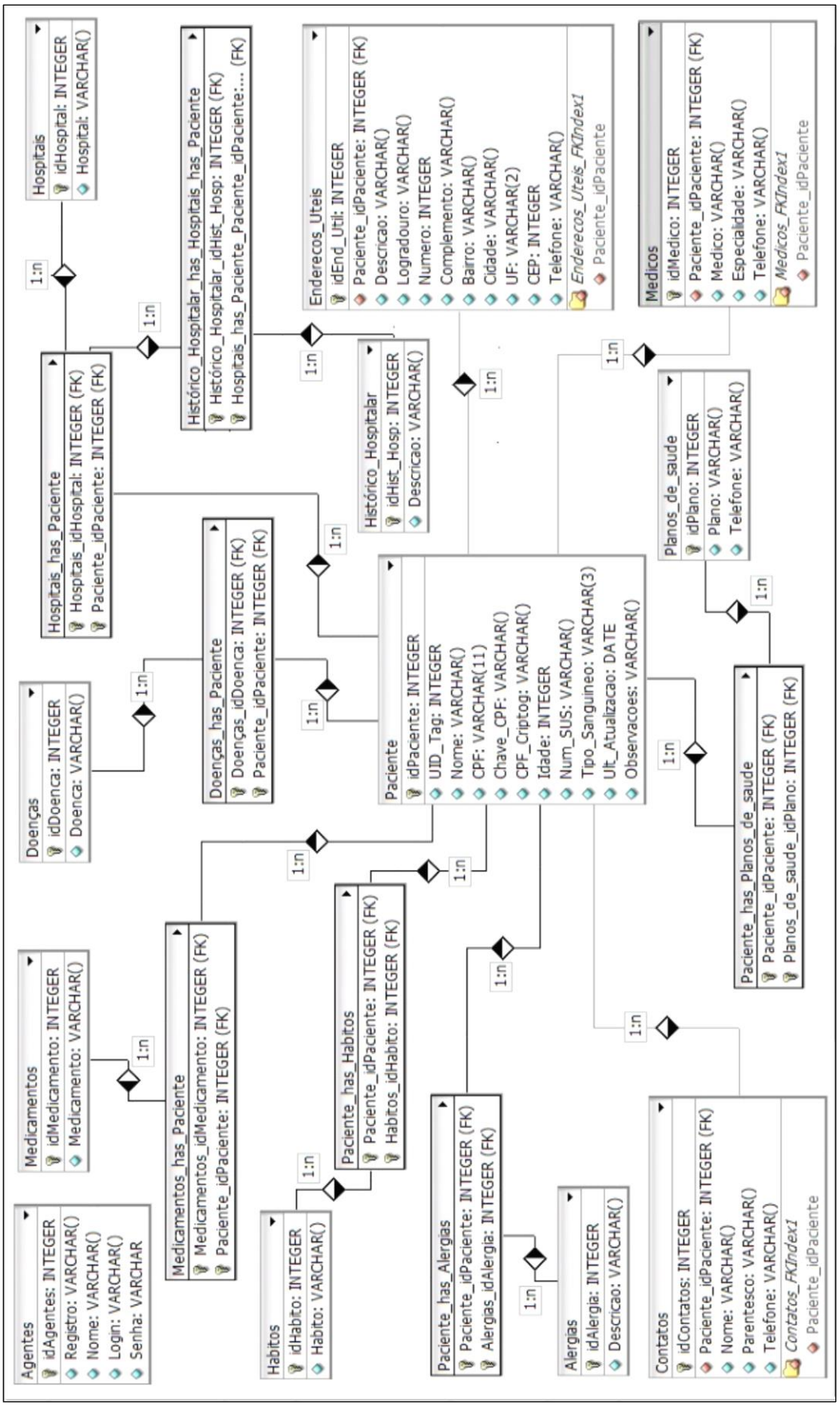


Figura 5.10: Diagrama ER
Fonte: do autor

No campo superior esquerdo da Figura 5.10, nota-se a tabela *Agentes*, possuindo como chave primária *idAgente*. Esta é a única tabela que não possui relacionamento com outra tabela do modelo. Todas as demais tabelas partem ou derivam da ligação com a tabela *Pacientes*, onde estes relacionamentos serão detalhados a seguir.

Para observar a relação dos pacientes com os seus medicamentos, nota-se que existe a tabela *Paciente* com sua chave primária *idPaciente*. Na outra ponta, existe a tabela *Medicamentos* com sua chave primária *idMedicamentos*. Entre estas duas tabelas existe uma tabela terciária, possuindo uma chave composta, formada pelas chaves estrangeiras da tabela de medicamentos e da tabela de pacientes. Esta tabela é responsável pela ligação de um paciente com vários medicamentos, assim como a ligação de um medicamento com vários pacientes.

Quanto à relação dos pacientes com os suas doenças, percebe-se que existe a tabela *Paciente* com sua chave primária *idPaciente*. Na outra extremidade, existe a tabela *Doencas* com sua chave primária *idDoencas*. Entre estas tabelas, existe a tabela terciária, possuindo uma chave composta, formada pelas chaves estrangeiras da tabela de doenças e da tabela de pacientes. Esta tabela tem a função de realizar a ligação de um paciente com várias doenças, do mesmo modo como a ligação de uma doença com vários pacientes.

Na relação dos pacientes com os seus hospitais, observa-se que existe a tabela *Paciente* com sua chave primária *idPaciente*. Do outro lado, encontra-se a tabela *Hospitais* com sua chave primária *idHospital*. Intermediando estas duas tabelas, verifica-se a tabela terciária, possuindo uma chave composta, formada pelas chaves estrangeiras da tabela de hospitais e da tabela de pacientes. Esta tabela tem a finalidade de fazer a ligação de um paciente com vários hospitais, da mesma forma como a ligação de um hospital com vários pacientes.

Quanto ao relacionamento de pacientes com seus históricos hospitalares, nota-se que existe a tabela terciária de pacientes e hospitais com sua chave composta *idPaciente* e *idHospital*. Na outra ponta, nota-se a tabela *HistoricoHospitalar* com sua chave primária *idHist_Hosp*. Entre a tabela terciária de pacientes e hospital e a tabela de históricos hospitalares, existe outra tabela terciária, possuindo uma chave composta, formada pelas chaves estrangeiras da tabela terciária de pacientes e hospitais e da tabela de históricos hospitalares. Esta tabela tem o objetivo de efetuar a ligação de um paciente

e um hospital com vários históricos hospitalares, na qual um histórico hospitalar pertencerá a um paciente e um hospital.

No que tange a relação dos pacientes com seus endereços úteis, o modelo possui a tabela *Paciente* com sua chave primária *idPaciente* ligando-se a tabela *Enderecos_Uteis*, onde um paciente pode ter vários endereços úteis e um endereço útil pertence a um paciente.

Para a relação dos pacientes com os seus médicos, encontra-se a tabela *Paciente* com sua chave primária *idPaciente* ligando-se a tabela *Medicos*, na qual um paciente pode ter vários médicos e um médico pertencerá a um paciente.

Com relação a ligação dos pacientes com seus planos de saúde, verifica-se que de um lado a tabela *Paciente* com sua chave primária *idPaciente*. Do outro lado, encontra-se a tabela *Planos_de_saude* com sua chave primária *idPlano*. Entre estas duas tabelas existe a tabela terciária, tendo uma chave composta formada pelas chaves estrangeiras da tabela de planos de saúde e da tabela de pacientes. Esta tabela tem o propósito de ligar um paciente a vários planos de saúde, além de poder ligar um plano de saúde com vários pacientes.

No vínculo de pacientes com os seus contatos, encontra-se a tabela *Paciente* com sua chave primária *idPaciente* conectando-se à tabela *Contatos*, estabelecendo a relação de um paciente poder ter vários contatos e um contato pertencer a um paciente.

No relacionamento dos pacientes com suas alergias, nota-se numa extremidade a tabela *Paciente* com sua chave primária *idPaciente*. Na outra, observa-se a tabela *Alergias* com sua chave primária *idAlergia*. No meio destas tabelas contém uma tabela terciária, possuindo uma chave composta formada pelas chaves estrangeiras da tabela de alergias e da tabela de pacientes. Esta tabela tem a função de manter a ligação de um paciente com várias alergias, de igual forma, uma alergia pode estar ligada a vários pacientes.

Por fim, para estabelecer a ligação dos pacientes com seus hábitos, percebe-se num lado a tabela *Paciente* com sua chave primária *idPaciente*. No outro, verifica-se a tabela *Habitos* com sua chave primária *idHabito*. Entre estas tabelas há uma tabela terciária, contendo uma chave composta formada pelas chaves estrangeiras da tabela de hábitos e da tabela de pacientes. Esta tabela tem o objetivo de realizar a ligação de um

paciente com vários hábitos, da mesma forma como um hábito pode fazer a ligação com vários pacientes.

Neste capitulo foi possível abordar as diferentes facetas que se pode modelar para o desenvolvimento de um sistema. Inicialmente, foi possível ter a compreensão dos requisitos externos ao sistema, como o levantamento de premissas, descritas de forma hipotética, para um sistema que pudesse ser utilizado em larga escala, quem sabe, à nível nacional.

Ainda nos requisitos externos, pôde-se ter uma ideia básica de como seriam os processos de inclusão de um indivíduo ao sistema, a atualização de seus dados, pois informações de saúde são dados dinâmicos, assim como a sua exclusão do sistema. Junto a isso, foi possível explanar como que se caracterizariam os processos para a implantação do *chip* no corpo, assim como uma possível substituição ou até a sua remoção caso fosse necessário.

Quanto a modelagem do sistema em si, através dos recursos da UML e do modelo ER, pôde-se ter uma visão de como poderia se proceder o desenvolvimento deste sistema, de acordo com padrões consolidados internacionalmente, do qual foi possível observar os processos para definir uma visão geral do sistema quanto ao cadastramento e as consultas através dos casos de uso, definição de métodos e atributos por meio do diagrama de classes, a dinâmica de eventos do sistema através da elaboração do diagrama de sequência, além da definição da estrutura de banco de dados através do diagrama ER.

6. CRIPTOGRAFIA DO MODELO DO SISTEMA

AES – Rijndael

O algoritmo AES (*Advanced Encryption Standard*), se caracteriza por ser do tipo *block cipher*, ou seja, são algoritmos que utilizam criptografia simétrica, classificados em cifras de blocos, no qual realizam a encriptação das informações em blocos de tamanho fixo. O seu bloco possui tamanho de 128 bits utilizando-se de chaves que variam de 128, 192 e 256 bits. O NIST (Instituto Nacional de Padrões e Tecnologia dos EUA) o adotou como um padrão em novembro de 2001, demorando cerca de cinco anos para que concluísse o processo de padronização. Este algoritmo foi criado pelos criptógrafos belgas, Joan Daemen e Vicente Rijmen (DA COSTA CARMO; CORRÊA, 2009).

O processo de execução do AES exige alguns dados, entre eles a *S-Box*, que se trata de uma tabela de substituição estática, o estado, correspondente ao bloco de dados de entrada que sofre alterações conforme o algoritmo se transforma, a chave e por fim a chave de expansão, que representa uma versão modificada da chave (*Announcing The Federal* apud TREVISAN; DA SILVA SACCHI; SANABRIA, 2013).

Para o processo de cifragem, existem quatro transformações que o módulo do AES possui, sendo eles, o *SubBytes*, responsável pela substituição dos bytes do estado pelos bytes da *S-Box*, *ShiftRows*, que possui a finalidade de rotacionar ciclicamente as linhas do estado, para a segunda em uma casa, para a terceira em duas casas e para a quarta em três casas, a *MixColumns*, que corresponde a uma transformação dos dados das colunas do estado através da multiplicação por um polinômio irredutível fixado e por fim a *AddRoundKey*, que possui a função de mesclar as colunas com uma das chaves geradas pela rotina de expansão.

Para o processo de decifragem, todo o processo é feito ao contrário, utilizando as quatro transformações de modo inverso, além da *S-Box* usada pela transformação *SubBytes* também ser invertida, ou seja, cada uma dessas operações realizam o processo inverso de suas equivalentes no módulo de cifragem (TREVISAN; DA SILVA SACCHI; SANABRIA, 2013).

Uma das vantagens do AES é sua rapidez para ser implementada, tanto em nível de software quanto de hardware, além de necessitar de pouca memória. Este algoritmo

vem sendo bastante utilizado devido a ele ser um novo padrão de *block cipher*. (DA COSTA CARMO; CORRÊA, 2009).

Por apresentar estas vantagens, mostrando-se como uma tendência para os padrões de criptografia, optou-se pela sua adoção no modelo de sistema proposto. A ideia se trata de criptografar o CPF da pessoa, utilizando como chave o valor gerado por um algoritmo de *hash*, que consiste num algoritmo que mapeia dados de comprimento variável para dados com comprimento fixo (Wikipédia, 2014c), feito com o próprio CPF da mesma. O CPF criptografado seria gravado na *tag* de biovidro (com propriedade de gravação única, premissa importante para que se evite regravações, que poderiam ocorrer de forma indevida por terceiros).

No registro do banco de dados da pessoa fica armazenado o valor do CPF criptografado (referente à informação gravada na *tag*), a chave que foi gerada e o próprio CPF, que serve para teste de validação da descriptografia. O motivo de se criptografar o CPF seria pelo fato de impossibilitar seu reconhecimento por parte de terceiros, como numa tentativa de efetuar a leitura por algum tipo de leitor de forma indevida e sem o consentimento da pessoa. Dessa forma, somente o sistema em questão é que poderia identificar o indivíduo. Pensou-se em utilizar o CPF por ser o identificador único de cada pessoa, facilitando o uso deste modelo de identificação para outros sistemas, em trabalhos futuros, por exemplo.

Para a simulação da criptografia, foram realizados testes no software *CrypTool* 1.4.31, escolhendo a opção da criptografia pesquisada (*Encrypt/Decrypt* > *Symmetric* (*modern*) > AES (CBC)) utilizando-se a opção de chave de 128 bits, que foi também gerada neste software através do *Hash* MD5 128 bits (Indiv. Procedures > *Hash* > MD5).

Apesar de não ser possível a gravação da informação no *chip* a ser usado no protótipo do sistema, sendo o *chip* usado do tipo que somente permite leitura, além de possuir apenas 4 bytes que armazenam o seu código único, o estudo é válido, pois existem modelos de *chip* com dimensões similares à um grão de arroz, conforme pode ser observado na Figura 7.3 do capítulo Protótipo.

Estes modelos de *chip* são utilizados para o controle animal, que suportam funções de leitura e escrita e que possuem até 256 bytes de memória (modelo *Hitag*, por exemplo), mais que suficiente para guardar a informação criptografada de identificação. Para este caso, de acordo com o que foi pesquisado no mercado de RFID, como um dos

exemplos, o site da Microchip Brasil, especialista em produtos e soluções na identificação animal por RFID, a frequência utilizada é de 134,2KHz, diferente da frequência do *chip* e leitor a serem usados neste trabalho, que é 125KHz.

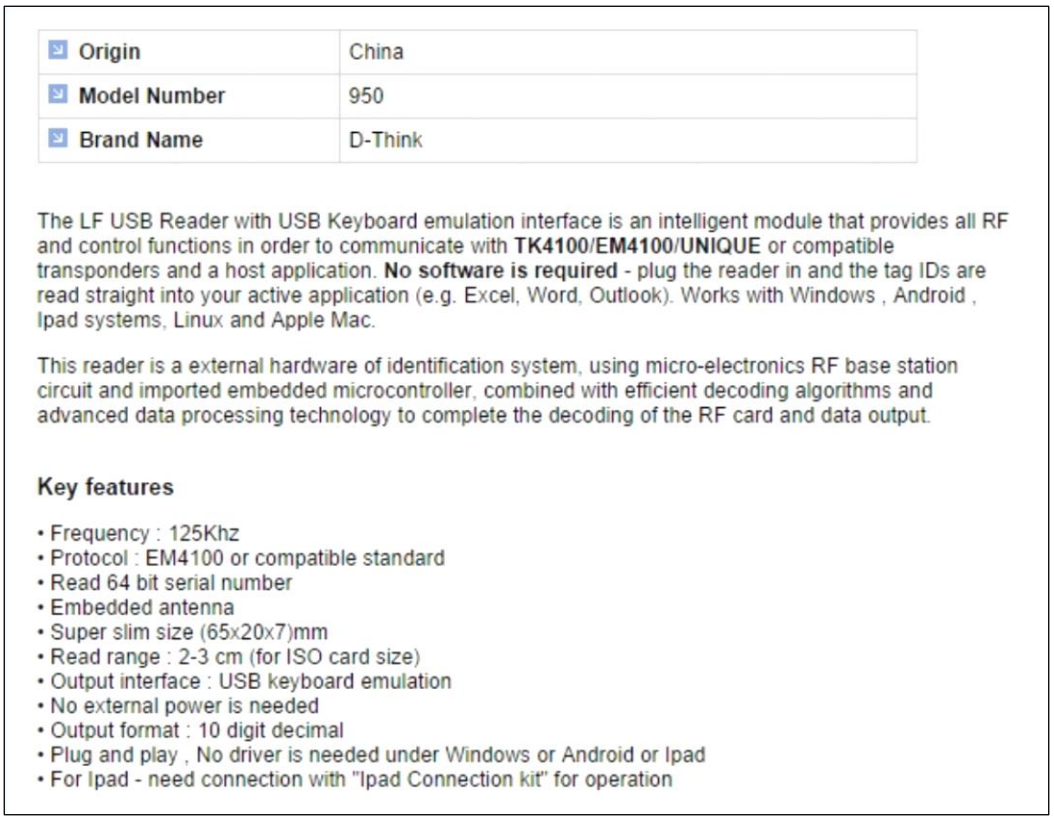
7. PROTÓTIPO

7.1 Verificação do meio de leitura

7.1.1 Descrição do leitor RFID

Na Figura 7.1 abaixo, pode-se observar as principais características do leitor a ser usado para o protótipo. Se trata de um leitor RFID da marca *D-Think*, modelo 950, de fabricação chinesa pela empresa *Guangzhou D-Think Technologies Inc*. Suas principais características são a frequência de 125KHz, comunicação com os padrões de protocolos TK4100/EM4100/UNIQUE, sendo o padrão EM4100 a ser utilizado para a leitura das *tags*, interface de conexão para a alimentação do micro leitor e saída de dados USB.

O formato de saída são 10 dígitos decimais, faz uso da tecnologia *Plug And Play*, dispensando instalação manual de drivers (Wikipédia, 2014a), é compatível com a maioria dos SO's do mercado, entre eles *Windows*, *Linux*, *Apple*, *iOS* e *Android*, sendo este último fundamental para o protótipo. A identificação da *tag* lida é transferida diretamente para a aplicação ativa do dispositivo em questão.



Origin	China
Model Number	950
Brand Name	D-Think

The LF USB Reader with USB Keyboard emulation interface is an intelligent module that provides all RF and control functions in order to communicate with **TK4100/EM4100/UNIQUE** or compatible transponders and a host application. **No software is required** - plug the reader in and the tag IDs are read straight into your active application (e.g. Excel, Word, Outlook). Works with Windows , Android , Ipad systems, Linux and Apple Mac.

This reader is a external hardware of identification system, using micro-electronics RF base station circuit and imported embedded microcontroller, combined with efficient decoding algorithms and advanced data processing technology to complete the decoding of the RF card and data output.

Key features

- Frequency : 125Khz
- Protocol : EM4100 or compatible standard
- Read 64 bit serial number
- Embedded antenna
- Super slim size (65x20x7)mm
- Read range : 2-3 cm (for ISO card size)
- Output interface : USB keyboard emulation
- No external power is needed
- Output format : 10 digit decimal
- Plug and play , No driver is needed under Windows or Android or Ipad
- For Ipad - need connection with "Ipad Connection kit" for operation

Figura 7.1: Especificações do leitor
Fonte: (*Guangzhou*, 2014)

7.1.2 Descrição da *tag*

Na Figura 7.2 abaixo, pode-se observar as principais características da *tag* a ser lida. Corresponde a um modelo de *tag* da marca LK, que opera na frequência de 125KHz, cujo tipo de *chip* contempla o padrão EM4100, que se trata do mesmo padrão que o leitor RFID do protótipo reconhece.

As dimensões do *chip* são de 2.12 x 12mm (por volta do tamanho de um grão de arroz), possuindo o material da caixa em biovidro para que seja compatível com organismo, além de conter um revestimento de *Parylene*, que se trata de uma substância antimigratório que impede a movimentação do *chip* após a implantação (Microchip Brasil, 2014).

Product Code	L-Link-A01	
Electronic	Operating Frequency	125 kHz
	Chip Type	EM4100/EM4305
	Memory	128 bit EEPROM
	Reading Distance	Dependent upon reader, environment and application
Physical	Dimensions	Φ2.12 x 12 mm
	Tagging Method	Subcutaneous
	Housing Material	Bio-glass
	Weight	0.03 g
Chemical and Mechanical Resistance	Water IP68	IP68, 68°F (20°C), 3.3 ft (1 m) x 24 h
	Withstands Exposure To	Alcohol, ammonium chloride 25%, fuel B, hydrochloric acid 10%, saltwater
	Environmental Test Conditions	68° F (20°C), 100 h
	Vibration	IEC 68.2.6 (10 g, 10 to 2000 Hz, 3 axis, 2.5 h)
	Shock	IEC 68.2.29 (40g, 18ms, 6 axis, 2000 times)
Thermal	Storage	-40° to +158°F (-40° to +70°C)
	Operating	-40° to +158°F (-40° to +70°C)
	Peak	248° F (120°C), 100 h; 284° F (140°C), 10 h
Other	Standards	ISO 11784/11785, ISO 14223
	Options	Parylene coating (when ordering coating, increase base model number by one), custom programming
	Warranty	2 Years

Figura 7.2: Especificações do *chip*
Fonte: (*AliExpress*, 2014)

Abaixo, na Figura 7.3, pode-se observar o hardware a ser utilizado para o protótipo, consistindo em um micro leitor RFID USB 125KHz e *tags* RFID 125KHz, nos formatos de cartão e cápsula:

Figura 7.3: Leitor e *tags* RFID 125Khz (cartão e cápsula)
Fonte: do autor

7.1.3 Comunicação do leitor com o dispositivo móvel

Para viabilizar a comunicação do leitor com o dispositivo móvel em questão é necessário utilizar um conector OTG, possível de ser adquirido na maioria dos estabelecimentos de componentes eletrônicos, assim como o dispositivo móvel deve suportar esta função, pois se trata de uma especificação que permite a conexão de dispositivos externos com um smartphone, por exemplo, no qual este último atua como um host, permitindo que outros dispositivos USB se anexem a ele (Wikipédia, 2014b).

Dessa forma, assim como um teclado poderia ser anexado ao dispositivo móvel, permitindo a transferência dos dados informados para a aplicação ativa, o mesmo deve ocorrer no processo de transferência dos dados lidos pelo leitor RFID. Na Figura 7.4 abaixo, um modelo de cabo OTG:

Figura 7.4: Modelo de cabo OTG
Fonte: do autor

7.2 Padrão EM4100

7.2.1 Principais características

Este padrão de *chip*, que será utilizado no protótipo, apresenta uma matriz de 64 bits de memória programada à laser, sendo do tipo somente leitura, opera em LF, numa faixa de frequência entre 100 a 150KHz, possui uma dimensão muito pequena, tornando-se conveniente para implantação, além de consumir pouca energia. (EM MICROCHIP, 2004).

7.2.2 Princípio do sistema

De acordo com a Figura 7.5, de forma simplificada, a bobina do *chip* envia o sinal com os dados modularizados, onde o oscilador do leitor, por meio de sua antena, capta o seu sinal modulado pelo *chip*, faz a demodulação dele, aplica os filtros para realizar o ajuste no sinal demodulado, decodifica este sinal em dados e, ao final, os apresenta. (EM MICROCHIP, 2004).

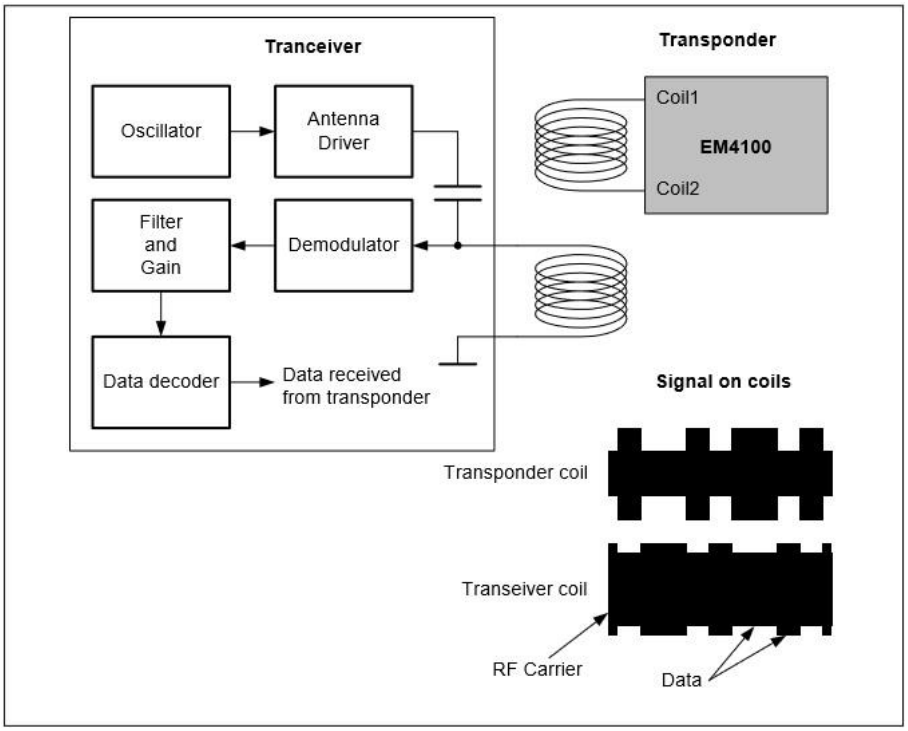


Figura 7.5: Princípio do sistema
Fonte: (EM MICROELECTRONIC, 2004)

7.2.3 Descrição Funcional

Na Figura 7.6, pode-se observar um detalhamento da estrutura que trata o sinal enviado pelo leitor até a devolução do mesmo. Pode-se observar que o sinal chega e em seguida passa pelo Retificador de Onda (*FULL WAVE RECTIFIER*), ocorrendo a conversão da tensão AC para DC por meio de uma ponte, limitando a tensão para evitar avarias internas.

Após aparece o Extrator de Clock (*CLOCK EXTRACTOR*), responsável por captar o sinal de um dos terminais da bobina (COIL1), utilizando-o para gerar o *clock* principal para a função lógica, onde a saída do Extrator de Clock controla o Sequenciador. O Sequenciador (*SEQUENCER*) fornece todos os sinais necessários para endereçar a matriz de memória e para codificar os dados seriais de saída.

Existem disponíveis três versões de máscaras programadas para a codificação da lógica, sendo elas a codificação Manchester, bifásica e PSK. O Sequenciador recebe o clock da bobina COIL1 por meio do Extrator de Clock e gera cada sinal interno controlando a memória (*MEMORY ARRAY*) e o codificador de dados (*DATA ENCODER*).

Depois de codificados os dados, o modulador de dados (*DATA MODULATOR*) é controlado pelo sinal do Controle de Modulação (*Control Modulation*), para induzir uma elevada corrente na bobina. Dependendo da versão de codificação de dados (Manchester, bifásico, ou PSK), pode ser usada somente um dos terminais da bobina (COIL2) ou ambas as bobinas (COIL1 e COIL2) para o envio do sinal modulado. (EM MICROCHIP, 2004).

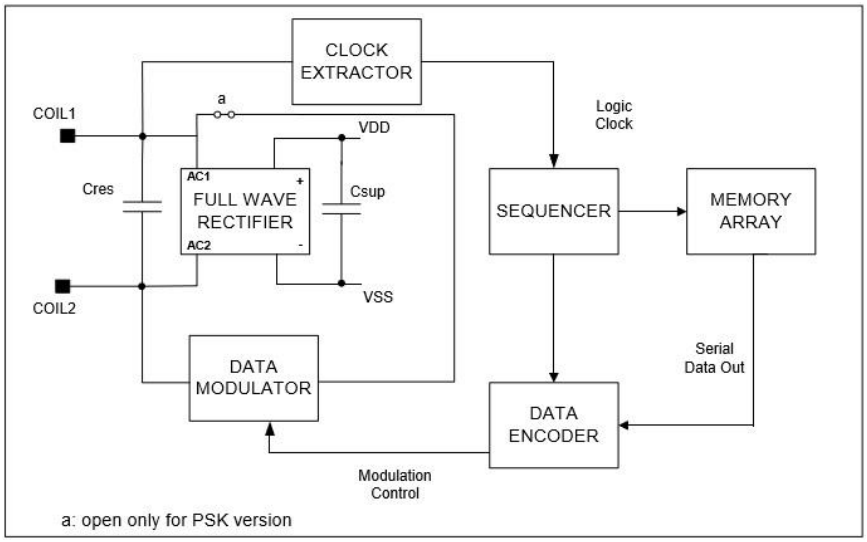


Figura 7.6: Composição interna do *chip*
Fonte: (EM MICROELECTRONIC, 2004

7.2.4 Matriz de memória

Conforme a Figura 7.7, o EM4100 contém 64 bits, divididos em cinco grupos de informações. São 9 bits usados para o cabeçalho, 10 bits de paridade das linhas (P0-P9), 4 bits de paridade das colunas (PC0-PC3), 40 bits de dados (D00-D93), e 1 bit de parada definido para lógica 0 (S0).

O cabeçalho é composto pelos nove primeiros bits, possuindo a máscara programada em "1", pois por questões de paridade e organização, estes dados não são disponibilizados na cadeia de dados. Os bits D00 à D03 e D10 à D13 são a identificação específica do cliente e os 32 bits D20 a D93 é onde fica armazenada a cadeia de dados que serão disponibilizadas pelo leitor, representando 10 dígitos. Ao todo são 64 bits emitidos serialmente em ordem para controlar o modulador, sendo esta sequência emitida continuamente até a energia ser desligada. (EM MICROCHIP, 2004).

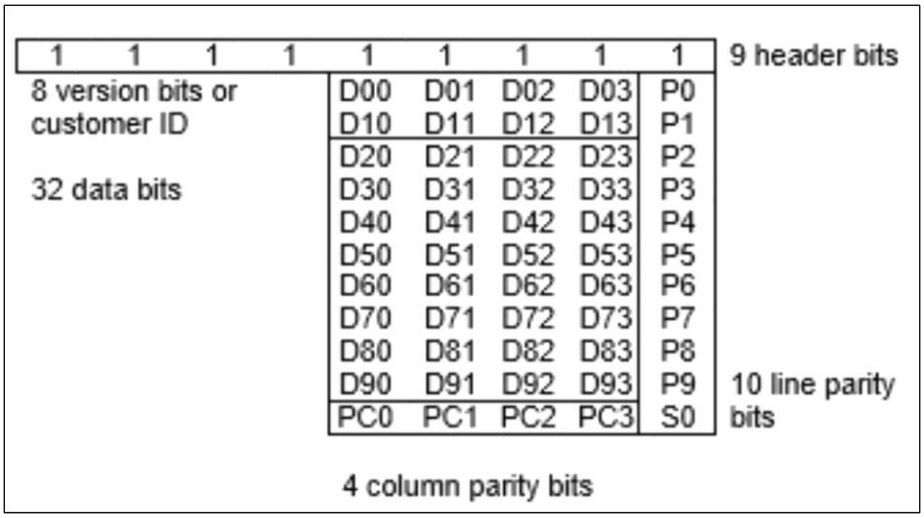


Figura 7.7: Matriz de memória
Fonte: (EM MICROELECTRONIC, 2004)

7.2.5 Descrição do código

Codificação Manchester

Para esse tipo, há sempre uma transição de ON para OFF ou de OFF para ON no meio do período de bit. A transição de bit lógico "1" para o bit lógico "0" ou bit lógico "0" para o bit lógico "1", ocorre na troca de fase, conforme se pode notar na Figura 7.8:

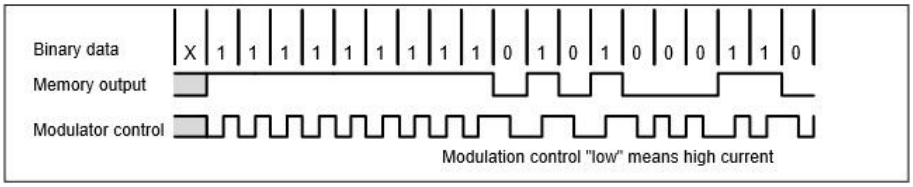


Figura 7.8: Esquema da codificação Manchester
Fonte: (EM MICROELECTRONIC, 2004)

Codificação Bifásica

Nesta codificação, no início de cada bit ocorrerá uma transição. Um bit lógico "1" irá manter o seu estado durante o período inteiro do bit e o bit lógico "0" irá mostrar uma transição no meio da duração do bit. Observe a Figura 7.9.

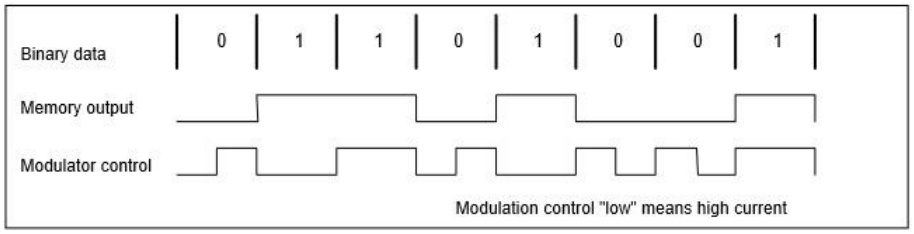


Figura 7.9: Esquema da codificação Bifásica
Fonte: (EM MICROELECTRONIC, 2004)

Codificação PSK

No PSK, o interruptor de modulação liga e desliga alternadamente a cada período da frequência portadora. Quando ocorre uma mudança de fase, um lógico "0" é lido da memória. Se não ocorrer mudança de fase depois de um ciclo de taxa de dados, um lógico "1" é lido, conforme é mostrado na Figura 7.10:

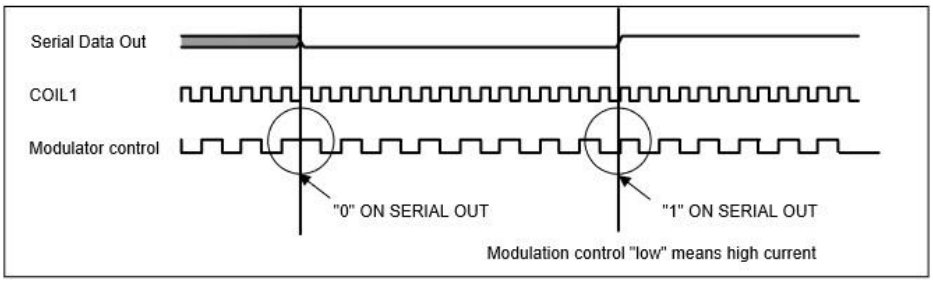


Figura 7.10: Esquema da codificação PSK
Fonte: (EM MICROELECTRONIC, 2004)

7.3 Sistema

7.3.1 Servidor

Para que seja possível tanto o cadastro de dados quanto a consulta dos mesmos neste protótipo do sistema, foi necessário configurar um servidor, neste caso, um servidor web. Como este sistema possui somente a finalidade de demonstrar exemplos de como funciona a sua dinâmica, dispensando-se a necessidade de configurações mais robustas quanto a nível de espaço, processamento, entre outros, optou-se por um serviço que pudesse disponibilizar um servidor web gratuito, portanto, com recursos mais modestos.

O serviço escolhido foi o *Free Web Hosting Area* (*http://freewebhostingarea.com/)*, que oferece os seus serviços desde 2005, disponibilizando hospedagens de diversos subdomínios, entre eles *orgfree.com*, *6te.net*, *ueuo.com*, entre outros. O subdomínio escolhido foi o *orgfree.com*, e o nome de domínio

escolhido foi “dadosmedicostcc”, formando a URL *http://www.dadosmedicostcc.orgfree.com*. Abaixo na Figura 7.11 pode-se observar:

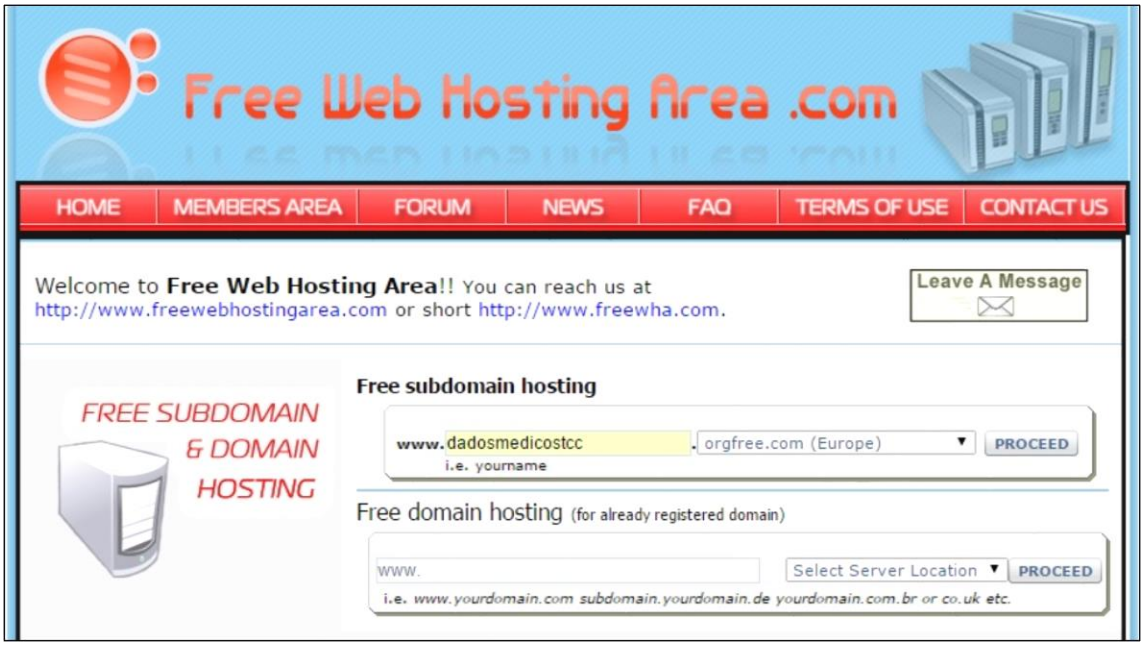


Figura 7.11: Tela do serviço *Free Web Hosting Area*
Fonte: (*http://freewebhostingarea.com*, 2014)

Neste domínio encontra-se o banco de dados, o registro do *webservice*, o *script* PHP que faz a comunicação entre o banco de dados e o *webservice*, as páginas HTML e *scripts* PHP para a realização do *login* dos agentes, para o cadastro destes e de pacientes.

Este serviço oferece os recursos e ferramentas básicas para colocar o servidor web deste protótipo de sistema em atividade, possuindo as seguintes características:

- Servidor *Apache* 2.2
- Servidores com processadores *octacore* com 32 GB de memória RAM por servidor
- Espaço de 1.5 GB, com limite de 12 MB por arquivo
- Tráfego ilimitado
- A conta nunca expira, exigindo somente um acesso mensal
- Suporte a vários formatos do PHP (*PHP sockets*, PHP XML, PHP SOAP, ...)
- Console para o controle do banco de dados *phpMyAdmin*
- Suporte completo para FTP
- Etc...

Na Figura 7.12 abaixo, é possível observar maiores detalhes das principais características oferecidas pelo serviço:

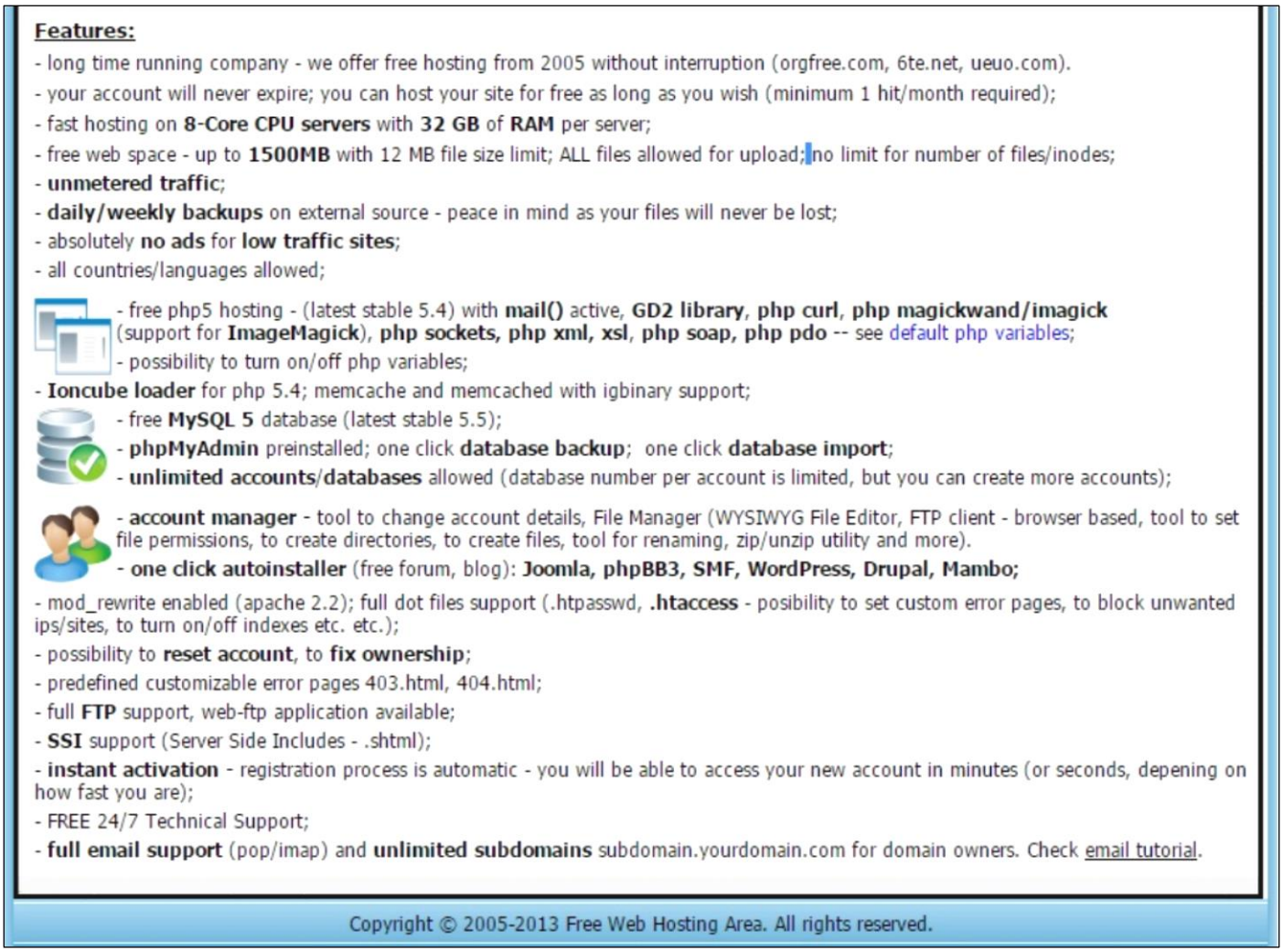


Figura 7.12: Principais características do *Free Web Hosting Area*
Fonte: (*http://freewebhostingarea.com*, 2014)

Como a linguagem utilizada para criar os *scripts* no servidor foi o PHP, optou-se para a criação do *webservice*, o uso de uma biblioteca que também fosse na mesma linguagem, no caso a biblioteca escolhida foi a *NuSoap* (*Nu Sphere PHP Web Services).*

Para as instruções de banco foi utilizada a linguagem SQL e o banco de dados usado foi o *MySQL 5*, disponibilizado pelo serviço do servidor. Para o desenvolvimento, as ferramentas utilizadas foram o XAMPP, que trata-se de um pacote de distribuição *Apache* com servidor, *MySQL*, PHP, *phpMyAdmin*, entre outros, sendo utilizado para testes numa máquina local, o *Notepad++*, que é um editor de texto que oferece suporte a várias linguagens de programação e o navegador *Chrome*, também para testes.

A implementação comentada das principais rotinas que controlam as consultas ao banco e ao *webservice*, assim como o registro deste e de seus métodos podem ser observados no APÊNDICE A.

7.3.2 Cadastramento

Antes do sistema ficar disponível para o acesso e consulta aos dados dos pacientes, é necessário realizar o cadastramento dos dados. Para isso, foi criado um protótipo de portal para que os agentes responsáveis pelo cadastramento dos dados possam realizar este trabalho.

Este protótipo consiste em três páginas, sendo a primeira delas a página que o agente efetua o seu *login* junto ao sistema, conforme mostra a Figura 7.13 abaixo:

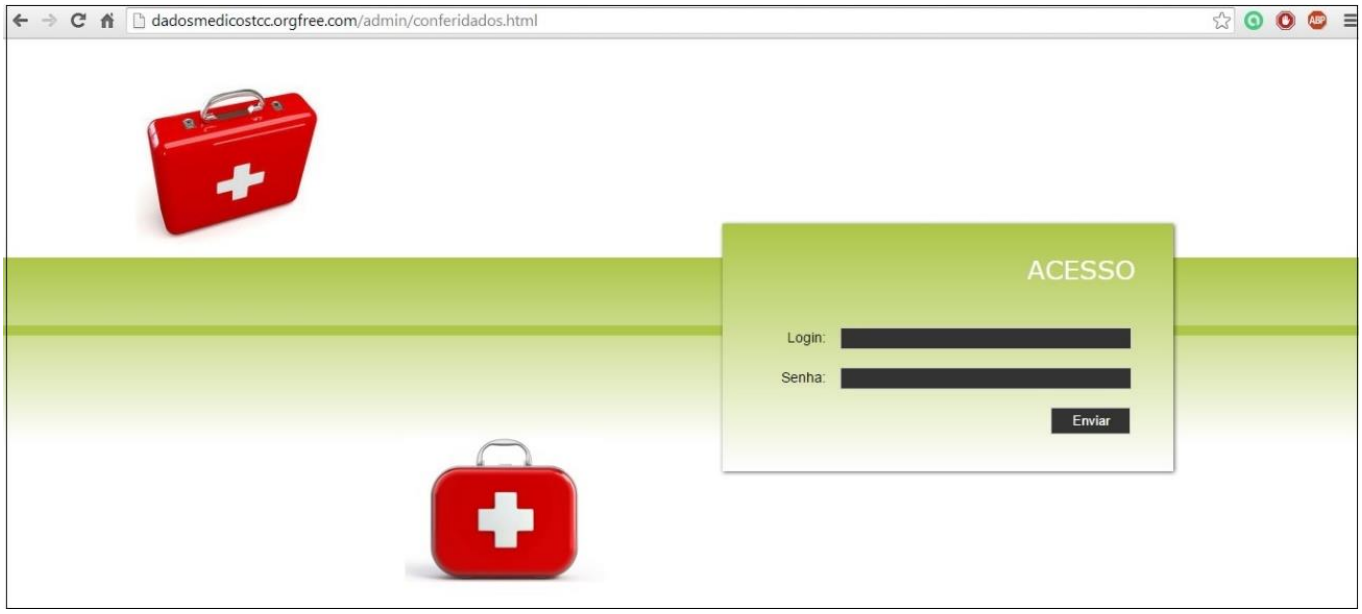


Figura 7.13: Tela de *login* para o cadastramento
Fonte: do autor

Após o agente autenticar-se ao sistema, este terá duas opções de cadastros. O primeiro deles se refere ao cadastramento de agentes, no caso deste protótipo, possui somente o usuário e senha, encontrando-se as opções de cadastrar, atualizar ou excluir. Abaixo do cadastro, encontra-se uma listagem dos agentes, contendo os principais dados de identificação para possibilitar a edição ou exclusão de um agente, de acordo como mostra abaixo a Figura 7.14:

Figura 7.14: Tela de Cadastro de Agentes
Fonte: do autor

A outra opção de cadastramento refere-se ao cadastro dos dados médicos dos pacientes, constando todos os campos definidos no capítulo Informações do Prontuário Médico, categorizados pela sua classificações (Dados Identificadores, Dados Vitais e Dados Complementares). Nele, encontra-se disponível as opções de cadastrar, atualizar ou excluir, além de conter uma listagem abaixo do cadastro contendo os principais dados de identificação do paciente, possibilitando a sua edição ou exclusão. Abaixo pode ser observadas as Figura 7.15 e 7.16 de como definiu-se este cadastro:

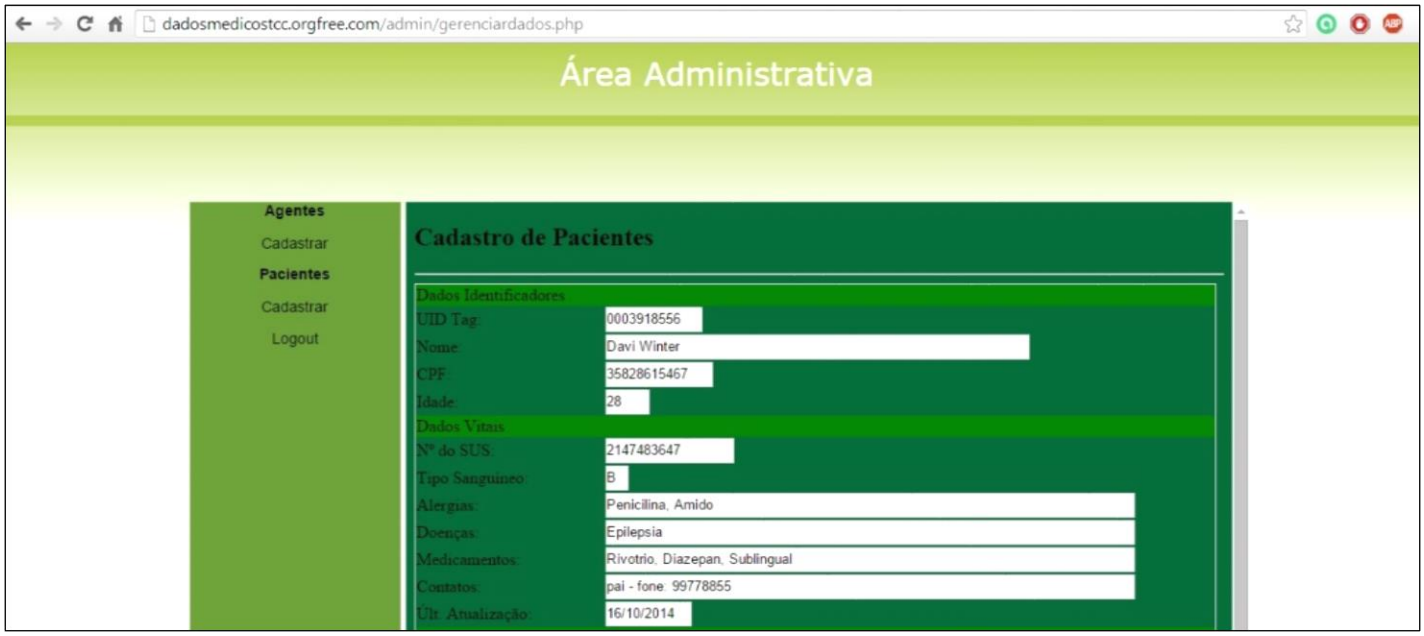


Figura 7.15: Tela de Cadastro de Pacientes – Parte 1
Fonte: do autor

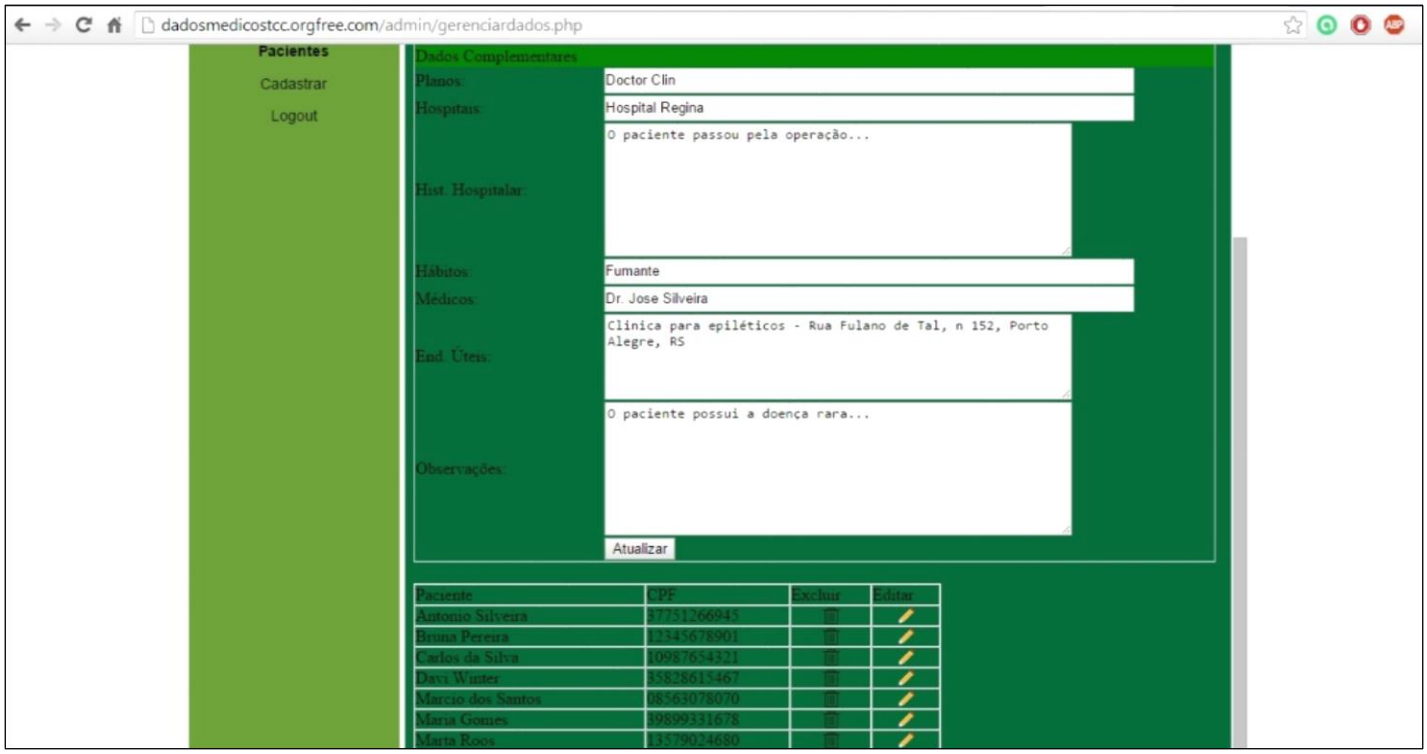


Figura 7.16: Tela de Cadastro de Pacientes – Parte 2
Fonte: do autor

O acesso ao protótipo do portal de cadastramento de agentes e dados médicos de pacientes pode ser conferido através do link *http://www.dadosmedicostcc.orgfree.com*, podendo ter acesso aos cadastros pelo *login* "agente" e senha "dadosmedicos" (por período indeterminado). Para a implementação deste portal fez-se uso das linguagens PHP, HTML, *JavaScript* e SQL. Quanto as ferramentas de apoio, foram utilizados o editor de texto *Notepad++* e o navegador *Chrome* para testes. O código comentado das principais rotinas deste portal pode ser verificado no APÊNDICE B.

7.3.3 Consulta

Por fim, esta é a etapa que representa o propósito principal do protótipo deste sistema, no qual um agente, via um dispositivo móvel *Android*, após autenticar-se com o sistema, realiza a consulta dos dados médicos de um determinado indivíduo, desde que este possua um *chip* implantado (para a leitura através do leitor RFID, característica principal para a identificação) ou informando-se o CPF desta pessoa.

Uma vez instalado o aplicativo no seu dispositivo *Android* e possuindo uma conexão com a internet, o agente em operação está apto para autenticar-se ao sistema por meio de seu *login*, conforme é mostrado na Figura 7.17:

Figura 7.17: Tela de *login* do usuário no *Android*
Fonte: do autor

Após o sistema validar o *login* do agente, este passa para a próxima etapa, que consiste na identificação do paciente no sistema, na qual contempla três opções de busca:

1. UID da *tag*: Trata-se do código fixo do *chip* (somente leitura), lido por um leitor RFID 125KHz;
2. CPF criptografado: Para um cenário onde possa ser lido um *chip* do tipo regravável por meio de um leitor RFID 125KHz, desde que suporte o armazenamento de um CPF criptografado nas especificações determinadas;
3. CPF: Opção em que o agente digita o CPF do paciente;

No protótipo deste aplicativo foi incluída uma caixa de seleção que associa alguns exemplos de CPF's ao seu valor criptografado, apenas para fins de demonstração da busca, pois a criptografia deste ocorre no cadastro e sua validação no processo de busca. A Figura 7.18 apresenta o layout desta segunda tela:

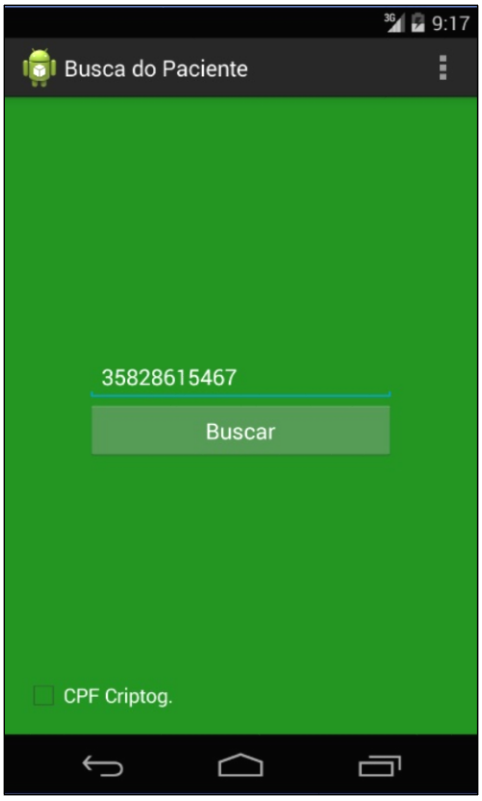


Figura 7.18: Tela de Busca do Paciente no *Android*
Fonte: do autor

Depois de informado um dos três parâmetros citados acima no campo de busca do paciente, basta o agente executar a busca e aguardar. Dentro de poucos segundos, caso a pessoa seja identificada, o aplicativo deve retornar os dados médicos do indivíduo para que possam ser consultados pelo agente, finalizando o ciclo do aplicativo na terceira e última tela, conforme apresentado na Figura 7.19:

Figura 7.19 Tela de Dados do Paciente no *Android*
Fonte: do autor

Os dados apresentados nessa tela segue a ordem definida no capítulo Informações do Prontuário Médico, bastando o agente rolar a tela até o final para visualizar todas as informações deste paciente. Para uma nova consulta, basta voltar para a tela anterior e informar uma nova identificação.

A implementação deste aplicativo utilizou a linguagem Java, com o uso da ferramenta *Android Studio*, que consiste numa IDE para desenvolvimento de aplicativos *Android*, criada pela Google e para a comunicação do aplicativo com o *webservice*, fez-se uso da biblioteca *ksoap2-android-assembly-3.3.0*. As principais rotinas deste aplicativo estão documentadas no APÊNDICE C.

8. PESQUISA DE ACEITAÇÃO

Esta pesquisa possui caráter informal, no qual não foi utilizado método cientifico para a sua aplicação, tampouco para a escolha de amostra, possuindo a finalidade de somente apresentar qual seria a aceitação do público em geral na adesão a esse tipo de sistema, em que um indivíduo teria que se submeter a uma intervenção cirúrgica para o implante de um *chip* em seu corpo, viabilizando a sua identificação.

No questionário existem somente duas perguntas. A primeira questiona se a pessoa permitiria que um *chip* fosse implantado em seu corpo, sendo descrita na pergunta como seria esse *chip* e a finalidade deste procedimento e a segunda representa algumas das possíveis opções que fizeram com que a pessoa optasse por rejeitar a implantação de um *chip* no seu corpo. Abaixo segue a formulação das perguntas:

1) "Você permitiria que lhe fosse implantado um *chip* (abaixo da pele), revestido em uma cápsula de biovidro e biocompatível no seu corpo, com a dimensão de um grão de arroz, com a finalidade de poder ser identificado somente para o sistema de saúde, onde possuísse um prontuário médico seu, dispensando qualquer tipo de documentação numa situação que porventura precisasse receber socorro, facilitando o atendimento de socorristas na obtenção de informações vitais suas por meio desta identificação, podendo preservar sua integridade física ou até mesmo salvar sua vida?

   [ ]sim [ ]não"

2) "Caso sua resposta tenha sido negativa, o que lhe levou a esta conclusão (marque ao menos uma opção):

   [ ] Por se tratar de um procedimento invasivo

   [ ] Por que deve ser um procedimento muito doloroso

   [ ] Medo que o implante deste *chip* possa me causar uma infecção

   [ ] Por que deve ser uma sensação incômoda ter um *chip* implantado

   [ ] Tenho medo de ser identificado por terceiros mal intencionados

   [ ] Por questões religiosas

   [ ] Outros motivos"

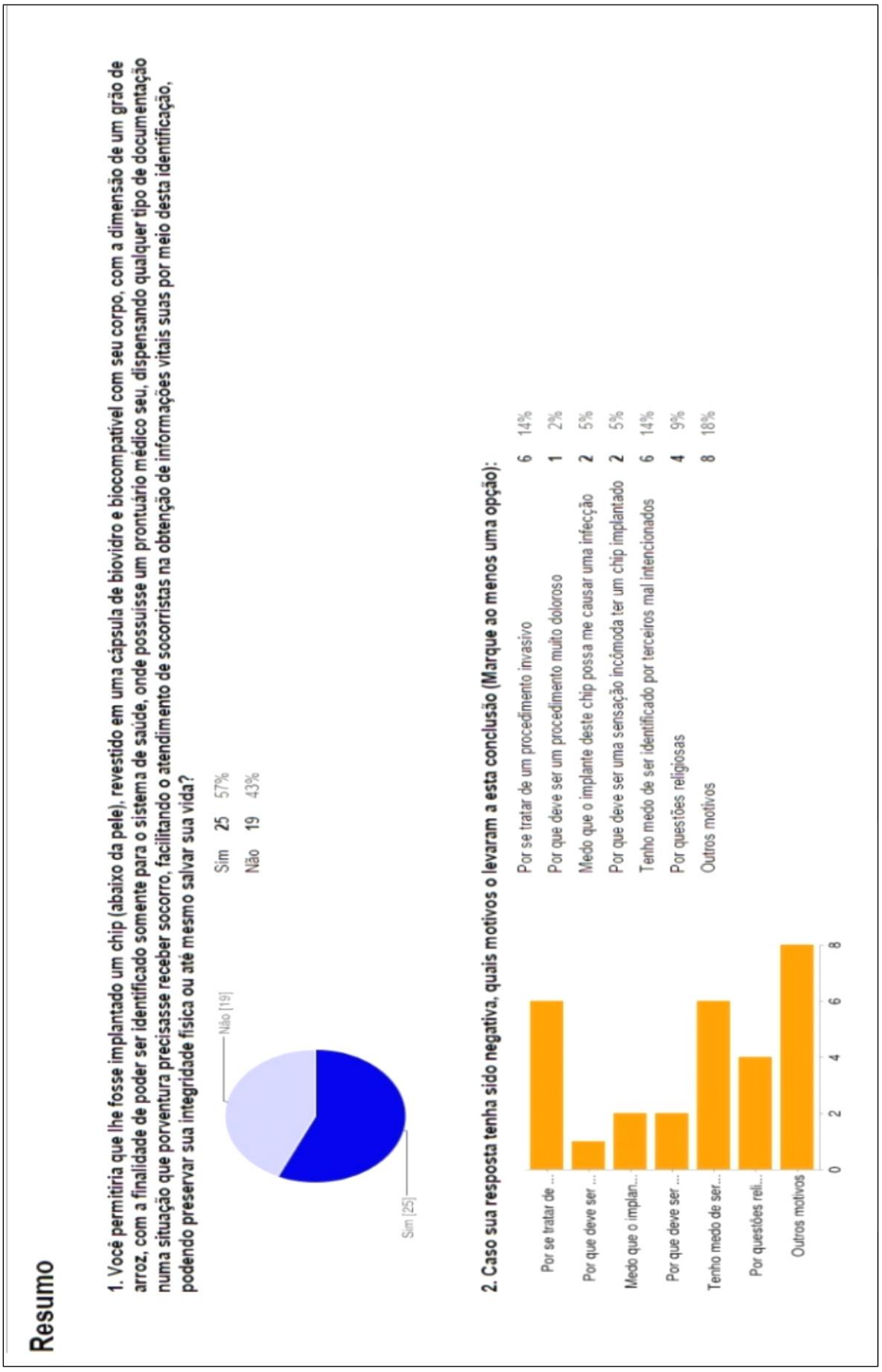


Figura 8.1: Resultado da Pesquisa
Fonte: Google Forms

Para a elaboração do questionário e apresentação do resultado foi utilizada a ferramenta *Google Forms*, podendo-se observar o resumo das respostas na Figura 8.1. A pesquisa foi disponibilizada através de mala direta por e-mail e redes sociais. O período da pesquisa foi de 29 de outubro de 2014 à 05 de novembro de 2014 e contou com 44 respostas.

Para esta pesquisa, a maior parte das respostas pendeu para a aprovação da implantação de um *chip* no corpo, sendo que 57% dos questionados aprovam a ideia do implante diante da circunstância descrita na pergunta e 43% reprovariam o implante, representando um equilíbrio de opiniões, neste caso, apresentando uma diferença de seis respostas entre um e outro.

Para os que reprovam a ideia de implante de *chip*, foram descritas algumas sugestões de motivos, no qual o motivo mais apontado foi por motivos diferentes dos sugeridos (Outros motivos), representando 18%, seguido de 14% pelo fato de ser um procedimento invasivo e também com 14% por terem medo de serem identificados por terceiros mal intencionados. Em terceiro lugar, 9% reprovam por questões religiosas, empatados com 5% ficaram os motivos de que o implante do *chip* possa causar alguma infecção e por ser uma sensação incômoda ter um *chip* implantado e por último, com 2%, por achar que deve ser um procedimento muito doloroso.

CONCLUSÕES

Diante dos estudos realizados sobre a RFID, se pôde formar uma ideia de como seria constituído parte do sistema proposto neste trabalho. Para a aplicação da etiqueta subcutaneamente à pele humana, verificou-se que o tipo mais adequado seria uma etiqueta passiva, visto a sua durabilidade, oferecendo mais resistência pela sua simplicidade, além de não possuir bateria, logo, minimizaria ou até dispensaria uma futura substituição, pois o ideal seria que esta etiqueta permaneça com o indivíduo ao longo de toda a sua vida.

Observou-se que a leitura de etiquetas injetáveis normalmente é feita em baixa frequência (faixa dos 125KHz e 134KHz) ou alta frequência (13,56 MHz), não encontrando-se casos em leituras de frequências superiores a estas. Como o objetivo é de que a leitura seja feita para dispositivos móveis, considerou-se que um leitor ideal para este cenário devesse ser pequeno e prático, operando numa destas frequências.

Por se tratarem de frequências que não consomem tanta energia, fez com que viabilizasse o seu fornecimento para a extração da informação da etiqueta pelo próprio dispositivo móvel.

Para o modelo de sistema proposto deste trabalho, verificou-se que existe, de forma direta ou indireta, a influência de todos os assuntos pertinentes à comunicação entre o cliente (dispositivo móvel) e um servidor web, contemplando os protocolos estudados, pois a comunicação precisa ser confiável (TCP/IP) e precisa do recipiente (HTTP) para a interação cliente-servidor.

Em relação as arquiteturas estudadas, foi necessário compreender como funciona a relação cliente-servidor, pois o dispositivo móvel, através de uma requisição com os dados do agente e do paciente precisará, via a identificação destes no servidor web, onde deverá tratar as solicitações, que podem ser múltiplas, de forma eficiente, disponibilizando sua resposta com os dados médicos do paciente a ser socorrido. O mesmo cenário se aplica no processo de atualização dos dados destes pacientes em uma base de dados neste servidor web.

Quanto aos WS, devido a sua independência de plataforma, pois facilita a integração de aplicações, surge como uma opção adequada para estruturar as informações na troca de dados entre cliente e servidor, sendo empregado tanto para o processo de atualização dos dados, em um *browser*, quanto na consulta destes pelo dispositivo móvel.

Quanto à plataforma *Android* do dispositivo móvel, todas as características analisadas se configuraram como essenciais para a elaboração do modelo de aplicativo *Android*, para desempenhar o lado cliente do sistema, sendo necessário o seu estudo para que se pudesse ter uma compreensão básica do funcionamento da arquitetura e componentes desta plataforma para possibilitar o desenvolvimento do aplicativo de consulta.

No que diz respeito a definição do prontuário médico de um indivíduo, os métodos e etapas realizados durante o processo de atendimento pré-hospitalar pelos profissionais de saúde, serviram de base para elaborar a composição dos dados do prontuário médico utilizado pelo modelo do sistema, complementados com outros dados pertinentes à identificação da pessoa, relativos aos sistemas de saúde utilizados por estas (como planos de saúde e hospitais) e para dados extras, que porventura podem não enquadrar-se nos dados definidos (observações).

Quanto a definição de requisitos e modelagem do sistema, apesar da definição dos processos abordados no trabalho mostrarem-se de certa forma “burocratizado”, esta modelagem é de suma importância quando se tem ideia de fazer um sistema que porventura tivesse uma grande abrangência no seu uso, envolvendo grandes volumes de dados e muitos acessos para o cadastramento e consulta de dados. Além do mais, possibilita um entendimento mais claro de como o sistema é constituído e de como se dará o seu funcionamento, assim como a adoção de boas práticas para a sua implementação.

Em relação a criptografia a ser usada no sistema, não foi possível aplicar a criptografia estudada no *chip* que conterá a identificação, pois o *chip* que foi possível adquirir neste presente trabalho se trata do tipo "*read only*", permitindo somente leitura de um código que já vem gravado de fábrica, além de, numa situação que fosse possível a gravação neste, possui ainda memória insuficiente para guardar um CPF criptografado, pois o *chip* do protótipo possui apenas 4 bytes (memória que armazena 10 dígitos) e para um CPF criptografado, de acordo com os testes no software *CrypTool*, necessitaria de ao menos 16 bytes.

No entanto, conforme descrito no capítulo 6, existem modelos de *chips* que suportam a função de gravação, que operam em baixa frequência e que possuem espaço suficiente para gravar uma identificação criptografada. Apesar disso, a criptografia estudada foi aplicada no sistema do protótipo, para os processos de cadastramento e

consulta, podendo ser utilizado, desde que o *chip*, o leitor e o dispositivo móvel consigam extrair o dado criptografado.

Tratando-se do protótipo do sistema, neste trabalho foi possível verificar as principais etapas necessárias para o seu desenvolvimento e aplicação, no qual foi pesquisado primeiramente o tipo de hardware que seria possível de se utilizar, respeitando-se pré-requisitos como uma das frequência a ser utilizada para a leitura de *chips* passíveis de serem implantados em pessoas, especialmente no sentido de tamanho e composição da *tag*.

Já para a leitura da *tag*, foi necessário que o leitor pudesse atender condições como a capacidade de ter sua energia fornecida por um dispositivo móvel, como um smartphone e que ao mesmo tempo pudesse ser suportado pelo dispositivo no que se refere a comunicação, papel realizado pelo fato do leitor oferecer compatibilidade com o sistema *Android*, dispensando instalação manual de drivers (*Plug And Play*) e através do auxílio de um cabo OTG para fazer a "ponte", ressaltando que o dispositivo móvel *Android* deve ser compatível com o OTG.

Também foi possível explanar em maiores detalhes como que ocorre os processos de comunicação determinados pelas especificações EM4100 utilizados pelo leitor e os *chips* do protótipo, sendo estudado o princípio do sistema, detalhando-se a sua funcionalidade, esquema de memória, codificação, entre outros.

Quanto ao sistema, pôde-se constatar as etapas realizadas para a configuração do servidor web, para viabilizar os processos de cadastramento e consultas do sistema, intermediadas pela implementação de um *webservice*. Constatou-se também o desenvolvimento de um portal para a realização dos processos de cadastramento e atualização dos dados, indispensável para viabilizar as consultas aos dados médicos de um paciente, oferendo o meio para que os dados possam estar sempre atualizados.

Por fim, para a execução do propósito principal deste trabalho, foi realizado o desenvolvimento do aplicativo *Android*, que através do auxílio da RFID, poderá um agente, após sua autenticação, realizar a identificação do indivíduo através da leitura do *chip* implantado no corpo deste, ou, em última instancia, informando-se manualmente o CPF do paciente e ao final, uma vez cadastrado e localizado o registro, deverá o aplicativo receber os dados desta pessoa, auxiliando o agente, no que diz respeito a tempo e eficiência durante o socorro prestado à vítima.

Foram realizados testes no que tange o cadastramento de dados pelo portal, por meio de um *desktop*, assim como realização de consultas pelo aplicativo do dispositivo *Android*, cujo protótipo apresentou um desempenho satisfatório, funcionando em quase todos os testes, com algumas exceções, decorrentes da instabilidade do servidor web, possivelmente pelo fato de ser gratuito, logo, menos robusto, e algumas falhas quanto à leitura das *tags* de cápsula, possivelmente causada pela distância entre o leitor e a *tag*.

Para trabalhos futuros, fica o desafio de colocar em prática a gravação da identificação criptografada no *chip*, no qual este deve também possuir memória suficiente para isso, além de permitir gravação única e que opere a baixa frequência, para poder ser energizado por um dispositivo móvel. Com isso, poderia se gravar um dado genérico a todas as pessoas, como o CPF, trazendo maior segurança para o sistema, além de possibilitar a sua expansão para outras áreas além da saúde.

Além do mais, poderiam ser estudadas outras maneiras de identificar um indivíduo, não somente através da tecnologia RFID, mas também através da biometria, por meio da leitura das digitais, da íris, da simetria da face, entre outros meios que peculiarizam cada indivíduo. Entretanto, estes métodos de identificação poderiam até operar de forma conjunta, por exemplo, fazendo com que a adesão a esse tipo de sistema por parte da população em geral fosse mais “aceito”, pois o modelo proposto deste trabalho representa um tabu para boa parte das pessoas, o que de certa forma comprometeria a sua real aplicação.

REFERÊNCIAS BIBLIOGRAFICAS

ALIEXPRESS. **Tag RFID vidro, EM4100, apenas de leitura, D2.12x12mm**. 2014. Disponível em: <http://pt.aliexpress.com/item/RFID-Glass-Tag-EM4100-Read-only-D3x15mm/1592973790.html?isOrig=true#extend> Acesso em: 02 set. 2014.

BASTOS, Débora Andrade; SILVA, Flávia Marques da. **Estudo e implementação de controladores para sistemas RFID**. 2010. Disponível em: <http://bdm.bce.unb.br/bitstream/10483/859/1/2007_D%C3%A9boraBastos_Fl%C3%A1viaSilva.pdf> Acesso em: 03 mai. 2014.

BERNARDO, Cláudio Gonçalves. **A tecnologia RFID e os benefícios da etiqueta inteligente para os negócios**. Revista Eletrônica Unibero de Iniciação Científica, São Paulo, 2004. Disponível em: <http://aulas-gelsimar.googlecode.com/svn/trunk/TOLER%C3%82NCIA_FALHAS/A_Tecnologia_RFID.pdf> Acesso em: 04 mai. 2014.

BRAGA, Alexandre Melo; DO NASCIMENTO, Erick Nogueira; RODRIGUES, Lucas. **Introdução à Segurança de Dispositivos Móveis Modernos–Um Estudo de Caso em Android**. Simpósio em Segurança da Informação e de Sistemas Computacionais. Sociedade Brasileira de Computação–SBC, v. 12, 2012. Disponível em: <http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/ceseg:2012-sbseg-mc2.pdf> Acesso em: 24 mai. 2014.

CASTRO, Mauro Nuno Barbosa de. **Desenvolvimento de um sistema de localização baseado em tecnologia RFID**. 2012. Disponível em: <http://repositorium.sdum.uminho.pt/handle/1822/27815> Acesso em: 07 mai. 2014.

COUTO, Eng. Agr. José Luiz Viana de. **Sinais Vitais**. Disponível em: < http://www.ufrrj.br/institutos/it/de/acidentes/sinais.htm> Acesso em: 11 out. 2014.

DA COSTA CARMO, Luiz Fernando Rust; CORRÊA, André Sion Fernandes Muniz. **Algoritmos Simétricos para Software Embarcado**. 2009. Disponível em: <http://equipe.nce.ufrj.br/rust/Mestrado%202009/SionSimetricAlg2009_SR.pdf> Acesso em: 10 set. 2014.

DE CF MACHADO, João Guilherme; NANTES, José Flávio Diniz. **Identificação eletrônica de animais por rádio-frequência (RFID): Perspectivas de uso na pecuária de corte**. Revista Brasileira de Agrocomputação, v. 2, n. 1, p. 29-36, 2004. Disponível em: <http://agrocomputacao.deinfo.uepg.br/junho_2004/Arquivos/RBAC_Artigo_04.pdf> Acesso em: 06 mai. 2014.

DE OLIVEIRA JUNIOR, Edson Alves; DE MATTOS FORTES, Renata Pontin. **Arquitetura de Software na Web Atual: Processamento no Servidor**. Disponível em: <http://www.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_ND_78.pdf> Acesso em: 17 mai. 2014.

DIAS, Renata Rampim de Freitas**. A importância de um middleware para o sistema RFID**. 2012. Disponível em: <http://brasil.rfidjournal.com/artigos/vision?9940>. Acesso em: 07 mai. 2014.

EM MICROELECTRONIC - MARIN SA. **EM4100** - **Read Only Contactless Identification Device**. 2004. Disponível em: <http://www.mikroe.com/downloads/get/1084/> Acesso em: 10 set. 2014.

GOMES, Hugo Miguel Cravo. **Construção de um sistema de RFID com fins de localização especiais**. 2007. Disponível em: <http://hdl.handle.net/10773/1876> Acesso em: 07 mai. 2014.

Guangzhou D-Think Technologies Inc. **EM4100&Compatibility RFID USB Disk Reader**. 2014. Disponível em: <http://dthinkrfid.en.ec21.com/EM4100_Compatibility_RFID_USB_Disk--6571154_6571211.html > Acesso em: 01 set. 2014.

GUEDES, Gilleanes T. A. **UML 2 - Uma abordagem prática**. 2. ed. São Paulo: Novatec Editora, 2011. 484 p.

HEUSER, Carlos Alberto. **Projeto de banco de dados**. Sagra Luzzatto, 2001.

JILOVEC, **Nahid. EDI, UCCNet & RFID: Synchronizing the supply chain**. Loveland: 29th Street Press, 2004. 288 p.

JÚNIOR, Marco Antônio PACHECO; DE OLIVEIRA CASTRO, Reinaldo**. Um estudo de caso da plataforma Android com Interfaces Adaptativas**. [s.d.]. Acessado em, v. 25, p. 19. Disponível em: <http://www.fgh.escoladenegocios.info/revistaalumni/artigos/Artigo_Marco%20Antonio.pdf> Acesso em: 22 mai. 2014.

LOPES, Carlos J. Feijó; RAMALHO, José Carlos. **Web Services: Metodologias de Desenvolvimento**. 2004. Disponível em: <http://hdl.handle.net/1822/559> Acesso em: 18 mai. 2014.

MACK, Roger Schneider. **Sistema de recomendação baseado na localização e perfil utilizando a plataforma android**. 2010. Disponível em: <http://hdl.handle.net/10183/28328> Acesso em: 20 mai. 2014.

MARTINS, Rafael JW de A. **Desenvolvimento de Aplicativo para Smartphone com a Plataforma Android**. 2009. Disponível em: <http://www.icad.puc-rio.br/~projetos/android/files/monografia.pdf> Acesso em: 20 mai. 2014.

MEIRELES, Artur Manuel Guedes**. Auxílio na Orientação de Invisuais Usando a Tecnologia RFID (SmartVision)**. 2009. Disponível em: <http://hdl.handle.net/10348/348> Acesso em: 05 mai. 2014.

Microchip Brasil. **Microchip Brasil > Home**. 2014. Disponível em: <http://microchipsbrasil.com.br/> Acesso em: 01 set. 2014.

OLIVEIRA, Alessandro de Souza; PEREIRA, Milene Franco. **Estudo da tecnologia de identificação por radiofrequência - RFID**. 2010. Disponível em: <http://bdm.bce.unb.br/bitstream/10483/829/1/2006_AlessandroeMilene.pdf> Acesso em: 05 mai. 2014.

OLIVEIRA, Maj. Marcos de. **Avaliação do Paciente no Ambiente Pré-hospitalar**. 2001. Disponível em: < http://www.aph.com.br/AV_PACIENTE_AMBIENTE_PH.htm> Acesso em: 11 out. 2014.

PASSARETTI, Caio Santi. RFID: **Identificação por radiofrequência movendo-se para o futuro**. 2010. Disponível em: <http://bdm.bce.unb.br/bitstream/10483/901/1/2008_CaioSantiPassaretti.pdf> Acesso em: 06 mai. 2014.

PINTO, Paulo Sergio Bernardo. **Webservice: a realidade da computação distribuída como ferramenta de integração entre tecnologias**. 2013. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/handle/1/826> Acesso em: 16 mai. 2014.

PRODANOV, Cleber Cristiano; FREITAS, Ernani Cesar de. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico**. 2. ed. Novo Hamburgo, RS: Feevale, 2013. 276 p. ISBN 9788577171583 Disponível em: <http://www.feevale.br/Comum/midias/8807f05a-14d0-4d5b-b1ad-1538f3aef538/E-book%20Metodologia%20do%20Trabalho%20Cientifico.pdf>. Acesso em: 15 mar. 2013

RODRIGUES, Marcus Vinicius Corrêa. **Segurança de Sistemas RFID com Modulação Aleatória**. 2010. Disponível em: <http://www.livrosgratis.com.br/arquivos_livros/cp142684.pdf> Acesso em: 05 mai. 2014.

TAVARES, Daniel Finger. **Middeware SOAP REST**. 2012. 72 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Feevale, Novo Hamburgo, 2012. Disponível em: <http://tconline.feevale.br/tc/files/0001_3294.pdf>. Acesso em: 19 maio 2014.

TEIXEIRA, Mário Antonio Meireles. **Suporte a serviços diferenciados em servidores web: modelos e algoritmos**. 2004. Tese de Doutorado. PhD thesis, ICMC-USP, São Carlos-SP. Disponível em: < http://www.researchgate.net/publication/34004100_Suporte_a_servios_diferenciados_em_servidores_web__modelos_e_algoritmos> Acesso em: 15 jun. 2014.

TREVISAN, Diogo Fernando; DA SILVA SACCHI, Rodrigo P.; SANABRIA, Lino. **Estudo do Padrão Avançado de Criptografia AES–Advanced Encryption Standard**. Revista de Informática Teórica e Aplicada, v. 20, n. 1, p. 13-24. 2013. Disponível em: <http://www.seer.ufrgs.br/index.php/rita/article/view/rita_v20_n1_p13/23763> Acesso em: 10 set. 2014.

UZEJKA, Guilherme de Moraes. **Modelando um cliente VoIP inteligente para a plataforma Android**. 2011. Disponível em: <http://hdl.handle.net/10183/31033> Acesso em: 20 mai. 2014.

Wikipédia, a enciclopédia livre. **Plug and Play**. 2014[a]. Disponível em: <http://pt.wikipedia.org/wiki/Plug_and_Play> Acesso em: 14 set. 2014.

Wikipédia, a enciclopédia livre. **USB On-The-Go**. 2014[b]. Disponível em: <http://pt.wikipedia.org/wiki/USB_On-The-Go> Acesso em: 14 set. 2014.

Wikipédia, a enciclopédia livre. **Hash**. 2014[c]. Disponível em: < http://pt.wikipedia.org/wiki/Hash> Acesso em: 28 out. 2014.

APÊNDICES

APÊNDICE A PRINCIPAIS ROTINAS DA IMPLEMENTAÇÃO DO SERVIDOR

```php
<?php

    $servidor="localhost"; //Nome do servidor (p/ Free Web Hosting Area é este mesmo)
    $usuario="878431";  //Nome de usuário (criado pelo servidor web)
    $senha="dadosmedicos"; //Senha do banco
    $banco="878431"; //Nome do banco (mesmo do usuario)

    //Conexão com o banco MySQL, passando por parâmetro o nome do servidor, usuário e senha
    $conn=mysql_connect($servidor_mysql, $usuario, $senha) or die("erro no código de conexão");

    //Seleção do banco, passando por parâmetro o nome do banco e a configuração da conexão
    mysql_select_db($banco, $conn) or die("erro no código de seleção do banco");


?>
```

Script de conexão com o banco de dados
Fonte: do autor

```php
<?php
    include('lib/nusoap.php'); //Inclusão da biblioteca NUSOAP
    include('admin/conexao.php'); //Inclusão do arquivo de conexão com o banco
    include("plugins/aes.class.php"); //Inclusão da biblioteca de criptografia AES

    //Instanciando o objeto NUSOAP que fará a configuração do webservice
    $servidor = new nusoap_server();
    //Configurações do webservice, passando o seu nome e o seu namespace
    $servidor->configureWSDL('dados_medicos', 'http://dadosmedicostcc.orgfree.com/');
    //Definições do namespace
    $nameSpace = 'http://dadosmedicostcc.orgfree.com/'; //'http://192.168.0.69/projetos/exemploWS/';
    $servidor->wsdl->schemaTargetNamespace = $nameSpace;

    //Função para descriptografar o CPF do paciente, contendo os parametros do CPF, chave do CPF e CPF
    //criptografado
    function descriptog_cpf($cpf, $chave_cpf, $cpf_criptog){
        //Instanciando o objeto de criptografia AES, passando a chave do CPF por parâmetro
        $aes=new AES($chave_cpf);
        //Guarda o resultado do CPF descriptografado com a função decrypt do objeto AES, sendo antes
        //decodificado pela função nativa do PHP "base64_decode", usada para possibilitar a transferência
        //do dado criptografado sobre a camada de transporte, possuindo em seu parametro o CPF criptografado
        $cpf_descriptog=$aes->decrypt(base64_decode($cpf_criptog));
        //Valida o CPF descriptografado se este for igual ao CPF registrado no banco
        if ($cpf_descriptog==$cpf){
            return true;
        }else{
            return false;
        }
    }
```

Script de comunicação com o *webservice* e de consultas ao banco – parte 1
Fonte: do autor

```php
//Função para autenticar agente no sistema
function autenticaUsuario($usuario, $senha){
    //Busca usuario credenciado no sistema de acordo com o seu login
    $consulta=mysql_query("select * from usuarios where usuario='$usuario' and senha='$senha'");
    $exibir=mysql_fetch_array($consulta);
    //Valida se o usuario foi localizado pelo seu login
    if (!empty($exibir["usuario"])) {
        return true;
    }else{
        return false;
    }
}
```

Script de comunicação com o *webservice* e de consultas ao banco – parte 2
Fonte: do autor

```php
//Função que realiza a busca do paciente e retorna os seus dados médicos, usando os três parametros
//de identificação (UID Tag, CPF ou CPF criptografado)
function buscaPaciente($uid_tag, $cpf, $cpf_criptog){
    //Busca pelo ID da tag, caso localizado, carrega os dados do paciente
    if (!empty($uid_tag) and empty($cpf) and empty($cpf_criptog)) {
        $consulta=mysql_query("select * from pacientes where uid_tag='$uid_tag'");
        $exibir=mysql_fetch_array($consulta);
        $achouPaciente=$exibir["cpf"];
    }

    //Busca pelo CPF do paciente, caso localizado, carrega os dados do paciente
    if (!empty($cpf) and empty($cpf_criptog) and empty($uid_tag)) {
        $consulta=mysql_query("select * from pacientes where cpf='$cpf'");
        $exibir=mysql_fetch_array($consulta);
        $achouPaciente=$exibir["cpf"];
    }

    //Busca pelo CPF criptografado do paciente, caso localizado, carrega os dados do paciente
    if (!empty($cpf_criptog) and empty($cpf) and empty($uid_tag)) {
        $consulta=mysql_query("select * from pacientes where cpf_criptog='$cpf_criptog'");
        $exibir=mysql_fetch_array($consulta);
        if ($exibir["cpf"]!="" and descriptog_cpf($exibir["cpf"], $exibir["chave_cpf"], $exibir["cpf_criptog"])){
            $achouPaciente=$exibir["cpf"];
        }else{
            $achouPaciente="";
        }
    }
```

Script de comunicação com o *webservice* e de consultas ao banco – parte 3
Fonte: do autor

```
        //Se o paciente foi localizado (com a identificação do seu CPF), retorna os seus dados
        //formatados por cada tipo de informação
        if (!empty($achouPaciente)) {
            $ult_atualizacao_eua=explode("-",$exibir["ult_atualizacao"]);
            return utf8_decode('* Dados Identificadores'."\n\n".
            '- UID Tag: '.$exibir["uid_tag"]."\n".
            '- Nome: '.$exibir["nome"]."\n".
            '- CPF: '.$exibir["cpf"]."\n".
            '- Idade: '.$exibir["idade"]."\n\n".
            '* Dados Vitais'."\n\n".
            '- Nº SUS: '.$exibir["num_sus"]."\n".
            '- Tipo Sanguíneo: '.$exibir["tipo_sanguineo"]."\n".
            '- Alergias: '.$exibir["alergias"]."\n".
            '- Doenças: '.$exibir["doencas"]."\n".
            '- Medicamentos: '.$exibir["medicamentos"]."\n".
            '- Contatos: '.$exibir["contatos"]."\n".
            '- Ult. Atualização: '.$ult_atualizacao_eua[2]."/".$ult_atualizacao_eua[1].
            "/".$ult_atualizacao_eua[0]."\n\n".
            '* Dados Complementares'."\n\n".
            '- Planos de Saúde: '.$exibir["planos"]."\n".
            '- Hospitais: '.$exibir["hospitais"]."\n".
            '- Hist. Hospitalares: '.$exibir["hist_hosp"]."\n".
            '- Hábitos: '.$exibir["habitos"]."\n".
            '- Médicos: '.$exibir["medicos"]."\n".
            '- End. Úteis: '.$exibir["end_uteis"]."\n".
            '- Observações: '.$exibir["observacoes"]);
        }else{
            return 'paciente desconhecido';
        }
    }
```

Script de comunicação com o *webservice* e de consultas ao banco – parte 4
Fonte: do autor

```
    //Registro no webservice do método "autenticaUsuario", possuindo os parametros de
    //usuario e senha e retorno se usuario foi ou não validado
    $servidor->register(
        'autenticaUsuario',
        array('usuario'=>'xsd:string',
              'senha'=>'xsd:string'),
        array('retorno'=>'xsd:boolean'),
        $nameSpace.'autenticaUsuario',
        $nameSpace.'autenticaUsuario',
        'rpc',
        'encoded',
        'Método para validar o usuário do sistema'
    );
    //Registro do método "buscaPaciente", possuindo os parametros do código da tag, CPF e CPF
    //criptografado e retorno dos dados médicos do paciente caso seja localizado
    $servidor->register(
        'buscaPaciente',
        array('uid_tag'=>'xsd:string',
              'cpf'=>'xsd:string',
              'cpf_criptog'=>'xsd:string'),
        array('retorno'=>'xsd:string'),
        $nameSpace.'buscaPaciente',
        $nameSpace.'buscaPaciente',
        'rpc',
        'encoded',
        'Método para buscar um paciente no sistema'
    );

    //Instruções necessárias para a comunicação com o webservice
    $HTTP_RAW_POST_DATA = isset($HTTP_RAW_POST_DATA) ? $HTTP_RAW_POST_DATA : '';
    $servidor->service($HTTP_RAW_POST_DATA);
?>
```

Script de comunicação com o *webservice* e de consultas ao banco – parte 5
Fonte: do autor

APÊNDICE B PRINCIPAIS ROTINAS DA IMPLEMENTAÇÃO DO PORTAL

```php
<?php
    include("../plugins/class.TemplatePower.php");//Inclusão de biblioteca que se comunica com o HTML
    include("conexao.php"); //Inclusão de arquivo para conexão com o banco

    $tpl=new TemplatePower("conferidados.html");
    $tpl->prepare();

    //Criação de variáveis para login e senha
    @$usuario=$_POST["usuario"];
    @$senha=$_POST["senha"];

    //Verifica se login e senha existe na tabela dos usuarios
    $consulta=mysql_query("select * from usuarios where LOWER(usuario)=LOWER('$usuario') and senha='$senha';")
              or die ("erro no codigo de consulta");

    //Extrair os campos da consulta
    $exibir=mysql_fetch_array($consulta);

    //Verificação que autentica o agente para o cadastro
    if(strtolower($usuario)==strtolower($exibir["usuario"]) and $senha==$exibir["senha"]) {
        //Inicia a variável de sessão
        session_start();
        //Cria variável de sessão
        $_SESSION["log"]=$exibir["usuario"];
        header("Location:gerenciardados.php");
    }else{
        //Sai da pasta adminbus e volta para a tela de login
        header("Location:conferidados.html");
    }
    //Imprime os dados na página
    $tpl->printToScreen();
?>
```

Implementação do *login* do agente
Fonte: do autor

```php
<?php
    include("../plugins/class.TemplatePower.php"); //Inclusão de plugin para linkar campos HTML com os dados do PHP
    include("conexao.php"); //Inclusão do arquivo de conexão com o banco
    $tpl=new TemplatePower("cadastro-usuario.html"); //Passando o formulário HTML para o template
    $tpl->prepare();

    //Criação das variáveis de usuário, senha e ação (cadastrar, editar, excluir)
    @$usuario=$_POST["usuario"];
    @$senha=$_POST["senha"];
    @$acao=$_REQUEST["acao"];

    $tpl->assign("textobotao","Cadastrar");

    if($acao=="Cadastrar"){
        //Grava as informações na tabela Usuarios
        if(!empty($usuario) and !empty($senha)){
            mysql_query("insert into usuarios
                        (usuario, senha)
                        values
                        ('$usuario', '$senha')") or die
                        ("Erro no código insert");

            $mensagem="Cadastro efetuado";

            echo"<script> alert('$mensagem');document.location=
            'cadastro-usuario.php';</script>";

            $tpl->assign("msg",$mensagem);
        }
    }
```

Implementação do cadastro do agente – parte 1
Fonte: do autor

```
// Criar uma listagem de usuários
@$consultaUsuarios=mysql_query("select * from usuarios order by id desc")
                    or die("erro no código da consulta de usuarios");
//Laço para imprimir os usuários na página do cadastro de usuários
while ($listaUsuarios=mysql_fetch_array($consultaUsuarios)){
    $tpl->newBlock("tabelaUsuarios");
    //Variaveis HTML para o resultado da consulta
    $tpl->assign("usuario",$listaUsuarios["usuario"]);
    $tpl->assign("id",$listaUsuarios["id"]);
}

$tpl->gotoBlock("_ROOT");

//Se a ação for igual a excluir, captura o registro escolhido (id) e o exclui
if($acao=="excluir"){
    $id=$_REQUEST["id"];
    //Excluir o registro
    mysql_query("delete from usuarios where id='$id';");
    $mensagem="Registro excluido";
    echo "<script>alert('$mensagem');document.location='cadastro-usuario.php'</script>";
}
```

Implementação do cadastro do agente – parte 2
Fonte: do autor

```
//Se a ação for igual a editar, captura o registro escolhido (id) e o busca no banco
if($acao=="editar"){
    $id=$_REQUEST["id"];
    $consultaReg=mysql_query("select * from usuarios where id='$id';")
                 or die("erro no código da consulta do usuário");
    $resConsulta=mysql_fetch_array($consultaReg);
    $tpl->assign("usuario",$resConsulta["usuario"]);
    $tpl->assign("senha",$resConsulta["senha"]);
    $tpl->assign("id",$resConsulta["id"]);
    $tpl->assign("textobotao","Atualizar");
}
//Realiza a edição no registro
if($acao=="Atualizar"){
    @$usuario=$_POST["usuario"];
    @$senha=$_POST["senha"];
    @$id=$_REQUEST["id"];

    mysql_query("update usuarios set usuario='$usuario',senha='$senha' where id='$id';")
    or die ("erro no código update");
    $mensagem="Dados atualizados";
    echo"<script> alert('$mensagem');document.location=
    'cadastro-usuario.php';</script>";
}


$tpl->printToScreen();
?>
```

Implementação do cadastro do agente – parte 3
Fonte: do autor

```php
<?php
    //Inclusão de biblioteca que se comunica com o HTML
    include("../plugins/class.TemplatePower.php");
    include("../plugins/aes.class.php"); //Inclusão de biblioteca de criptografia AES
    include("conexao.php");//Inclusão de arquivo para a conexão com o banco
    $tpl=new TemplatePower("cadastro-paciente.html");
    $tpl->prepare();

    //Criação das variáveis para guardar os dados do paciente
    @$uid_tag=$_POST["uid_tag"];
    @$cpf=$_POST["cpf"];
    @$nome=$_POST["nome"];
    @$idade=$_POST["idade"];
    @$num_sus=$_POST["num_sus"];
    @$tipo_sanguineo=$_POST["tipo_sanguineo"];
    @$alergias=$_POST["alergias"];
    @$doencas=$_POST["doencas"];
    @$medicamentos=$_POST["medicamentos"];
    @$contatos=$_POST["contatos"];
    @$ult_atualizacao=$_POST["ult_atualizacao"];
    @$ult_atualizacao_eua=explode("/",$ult_atualizacao);
    @$planos=$_POST["planos"];
    @$hospitais=$_POST["hospitais"];
    @$hist_hosp=$_POST["hist_hosp"];
    @$habitos=$_POST["habitos"];
    @$medicos=$_POST["medicos"];
    @$end_uteis=$_POST["end_uteis"];
    @$observacoes=$_POST["observacoes"];
```

Implementação do cadastro do paciente – parte 1
Fonte: do autor

```php
//Função que aplica a criptografia no CPF do paciente, passando por parametro seu CPF
function criptografaCpf($cpf){
    //Declara como variável global
    global $chave_cpf;
    //Aplica o código hash MD5 (função nativa do PHP), de 128 bits
    $chave_cpf=md5($cpf);
    //Instancia um objeto para a criptografia AES, possuindo como parametro a chave do CPF
    $aes=new AES($chave_cpf);
    //Declara como variável global
    global $cpf_criptog;
    //Guarda o CPF criptografado com a função "encrypt" do objeto AES sobre o parametro do
    //CPF do paciente, sendo aplicado encima a função nativa do PHP "base64_encode" (esta
    //ultima para possibilitar a transferencia do dado criptografado sobre a camada de transporte)
    $cpf_criptog=base64_encode($aes->encrypt($cpf));
}
```

Implementação do cadastro do paciente – parte 2
Fonte: do autor

```
@$acao=$_REQUEST["acao"];

$tpl->assign("textobotao","Cadastrax");

if($acao=="Cadastrax"){
    //Gravar as informações na tabela Usuarios
    if(!empty($nome) and !empty($cpf)){
        criptografaCpf($cpf);
        mysql_query("insert into pacientes
                    (uid_tag, cpf, nome, idade, num_sus, tipo_sanguineo, alergias, doencas,
                    medicamentos, contatos, ult_atualizacao, planos, hospitais, hist_hosp, habitos,
                    medicos, end_uteis, observacoes, chave_cpf, cpf_criptog)
                    values
                    ('$uid_tag', '$cpf', '$nome', '$idade', '$num_sus', '$tipo_sanguineo', '$alergias',
                    '$doencas', '$medicamentos', '$contatos', '$ult_atualizacao_eua[2].
                    $ult_atualizacao_eua[1].$ult_atualizacao_eua[0]', '$planos', '$hospitais',
                    '$hist_hosp', '$habitos', '$medicos', '$end_uteis', '$observacoes', '$chave_cpf',
                    '$cpf_criptog');") or die ("Erro no código insert");

        $mensagem="Cadastro efetuado";

        echo"<script> alert('$mensagem');document.location=
        'cadastro-paciente.php';</script>";

        $tpl->assign("msg",$mensagem);
    }
}
```

Implementação do cadastro do paciente – parte 3
Fonte: do autor

```
// Criar uma listagem de pacientes
@$consultaPacientes=mysql_query("select * from pacientes order by nome")
                    or die("erro no código da consulta de pacientes");
//Laço para imprimir os pacientes na página do cadastro de pacientes
while ($listaPacientes=mysql_fetch_array($consultaPacientes)){
    $tpl->newBlock("tabelaPacientes");
    //Variaveis HTML para o resulta da consulta
    $tpl->assign("cpf",$listaPacientes["cpf"]);
    $tpl->assign("nome",$listaPacientes["nome"]);
    $tpl->assign("id_paciente",$listaPacientes["id_paciente"]);
}
$tpl->gotoBlock("_ROOT");

//Se a ação for igual a excluir, captura o registro escolhido (id) e o exclui
if($acao=="excluir"){
    $id_paciente=$_REQUEST["id_paciente"];
    //Excluir o registro
    mysql_query("delete from pacientes where id_paciente='$id_paciente';");
    $mensagem="Registro excluido";
    echo "<script>alert('$mensagem');document.location='cadastro-paciente.php'</script>";
}
```

Implementação do cadastro do paciente – parte 4
Fonte: do autor

```php
// Se a ação for igual a editar, captura o registro escolhido (id) e o busca no banco
if($acao=="editar"){
    $id_paciente=$_REQUEST["id_paciente"];
    $consultaReg=mysql_query("select * from pacientes where id_paciente='$id_paciente';")
                or die("erro no código da consulta do paciente");
    $resConsulta=mysql_fetch_array($consultaReg);
    // Separar campos na variável
    $tpl->assign("uid_tag",$resConsulta["uid_tag"]);
    $tpl->assign("nome",$resConsulta["nome"]);
    $tpl->assign("cpf",$resConsulta["cpf"]);
    $tpl->assign("idade",$resConsulta["idade"]);
    $tpl->assign("num_sus",$resConsulta["num_sus"]);
    $tpl->assign("tipo_sanguineo",$resConsulta["tipo_sanguineo"]);
    $tpl->assign("alergias",$resConsulta["alergias"]);
    $tpl->assign("doencas",$resConsulta["doencas"]);
    $tpl->assign("medicamentos",$resConsulta["medicamentos"]);
    $tpl->assign("contatos",$resConsulta["contatos"]);
    $ult_atualizacao_eua=explode("-",$resConsulta["ult_atualizacao"]);
    $tpl->assign("ult_atualizacao",$ult_atualizacao_eua[2]."/".$ult_atualizacao_eua[1]."/".
    $ult_atualizacao_eua[0]);
    $tpl->assign("planos",$resConsulta["planos"]);
    $tpl->assign("hospitais",$resConsulta["hospitais"]);
    $tpl->assign("hist_hosp",$resConsulta["hist_hosp"]);
    $tpl->assign("habitos",$resConsulta["habitos"]);
    $tpl->assign("medicos",$resConsulta["medicos"]);
    $tpl->assign("end_uteis",$resConsulta["end_uteis"]);
    $tpl->assign("observacoes",$resConsulta["observacoes"]);

    $tpl->assign("id_paciente",$resConsulta["id_paciente"]);
    $tpl->assign("textobotao","Atualizar");
}
```

Implementação do cadastro do paciente – parte 5
Fonte: do autor

```php
//Ao atualizar, captura os dados informados no formulário HTML
if($acao=="Atualizar"){
    @$uid_tag=$_POST["uid_tag"];
    @$cpf=$_POST["cpf"];
    @$nome=$_POST["nome"];
    @$idade=$_POST["idade"];
    @$num_sus=$_POST["num_sus"];
    @$tipo_sanguineo=$_POST["tipo_sanguineo"];
    @$alergias=$_POST["alergias"];
    @$doencas=$_POST["doencas"];
    @$medicamentos=$_POST["medicamentos"];
    @$contatos=$_POST["contatos"];
    @$ult_atualizacao=$_POST["ult_atualizacao"];
    @$planos=$_POST["planos"];
    @$hospitais=$_POST["hospitais"];
    @$hist_hosp=$_POST["hist_hosp"];
    @$habitos=$_POST["habitos"];
    @$medicos=$_POST["medicos"];
    @$end_uteis=$_POST["end_uteis"];
    @$observacoes=$_POST["observacoes"];

    @$id_paciente=$_REQUEST["id_paciente"];
    //Criptografa o CPF do paciente
    criptografaCpf($cpf);
```

Implementação do cadastro do paciente – parte 6
Fonte: do autor

```
        //Realiza a edição no registro
        mysql_query("update pacientes set
                            uid_tag='$uid_tag',
                            cpf='$cpf',
                            nome='$nome',
                            idade='$idade',
                            num_sus='$num_sus',
                            tipo_sanguineo='$tipo_sanguineo',
                            alergias='$alergias',
                            doencas='$doencas',
                            medicamentos='$medicamentos',
                            contatos='$contatos',
                            ult_atualizacao='$ult_atualizacao_eua[2].$ult_atualizacao_eua[1].
                            $ult_atualizacao_eua[0]',
                            planos='$planos',
                            hospitais='$hospitais',
                            hist_hosp='$hist_hosp',
                            habitos='$habitos',
                            medicos='$medicos',
                            end_uteis='$end_uteis',
                            observacoes='$observacoes',
                            chave_cpf='$chave_cpf',
                            cpf_criptog='$cpf_criptog'
                            where id_paciente='$id_paciente';") or die ("erro no código update");

        $mensagem="Dados atualizados";
        echo"<script> alert('$mensagem');document.location=
        'cadastro-paciente.php';</script>";
    }
    //Imprime os dados na página HTML
    $tpl->printToScreen();
?>
```

Implementação do cadastro do paciente – parte 7
Fonte: do autor

APÊNDICE C PRINCIPAIS ROTINAS DA IMPLEMENTAÇÃO DA CONSULTA AOS DADOS MÉDICOS PELO *ANDROID*

```
public class DadosMedicosActivity extends Activity {

    public static final String SOAP_ACTION = "http://dadosmedicostcc.orgfree.com/autenticaUsuario";
    //"http://192.168.0.69/projetos/exemploWS/autenticaUsuario";
    public static final String URL = "http://dadosmedicostcc.orgfree.com/servidor.php?wsdl";
    //"http://192.168.0.69/projetos/exemploWS/servidor.php?wsdl";
    public static final String NAMESPACE = "http://dadosmedicostcc.orgfree.com";
    //"http://192.168.0.69/projetos/exemploWS";
    public static final String AUTENTICA_USUARIO = "autenticaUsuario";
    public static final String BUSCA_PACIENTE = "buscaPaciente";
    TextView tv;
```

Implementação da *activity* principal (tela de *login* do agente) – parte 1
Fonte: do autor

```
    //Função que chama o método do webservice "autenticaUsuario"
    public Boolean verificaUsuario(String nome, String senha){
        try {
            //Instanciando o objeto SOAP, passando por parâmetro o domínio e o método a ser chamado
            SoapObject autenticaUsuario = new SoapObject(NAMESPACE, AUTENTICA_USUARIO);
            //Adicionando as propriedades "usuario" e "senha" ao objeto SOAP
            autenticaUsuario.addProperty("usuario", nome);
            autenticaUsuario.addProperty("senha", senha);
            //Instanciando o envelope SOAP
            SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
            //Adicionando o objeto SOAP ao envelope
            envelope.setOutputSoapObject(autenticaUsuario);
            envelope.dotNet = true;
            //Instanciando um objeto HTTP, passando a URL do webservice
            HttpTransportSE http = new HttpTransportSE(URL);
            //Realizando a requisição http, passando o endereço do método do webservice e o envelope SOAP
            http.call(SOAP_ACTION, envelope);
            //Capturando em bytes (no formato UTF-8) o retorno do webservice
            byte array[] = envelope.getResponse().toString().getBytes("UTF-8");
            //Transformando o array de bytes em texto
            String resultString = new String(array, "UTF-8");
            //Verifica se o agente foi autenticado pelo webservice
            if (resultString.equals("true")){
                return true;
            }else{
                return false;
            }
        //Tratamentos de erro...
        } catch (SoapFault soapFault) {
            soapFault.printStackTrace();
```

Implementação da *activity* principal (tela de *login* do agente) – parte 2
Fonte: do autor

```
    private EditText edtUsuario;
    private EditText edtSenha;

    public void validaLogin (View view) throws SQLException, InterruptedException {
        Button b =(Button)view;

        edtUsuario = (EditText)findViewById(R.id.editTextUsuario);
        edtSenha = (EditText)findViewById(R.id.editTextSenha);
        String mensagem = "";

        //Sai da função caso os campos de usuário ou senha estejam vazios
        if (edtUsuario.length() == 0 || edtSenha.length() == 0) {
            return;
        }
        //Se o agente foi autenticado, chama o método que instancia a tela para a busca do paciente
        if (verificaUsuario(edtUsuario.getText().toString(), edtSenha.getText().toString()) == true) {
            TelaBuscarPaciente(R.layout.activity_buscar_paciente);
        }else{ //Senao mostra alerta de usuário/senha incorreto
            mensagem = "Usuário/Senha incorreto(s)";
            android.app.AlertDialog alertDialog;
            alertDialog = new AlertDialog.Builder(this).create();
            alertDialog.setTitle("Login");
            alertDialog.setMessage(mensagem);
            alertDialog.show();
        }
    }
```

Implementação da *activity* principal (tela de *login* do agente) – parte 3
Fonte: do autor

```
    //Instanciando a tela de busca do paciente
    public void TelaBuscarPaciente (int view){
        Intent it = new Intent(this, BuscarPacienteActivity.class);
        startActivity(it);
    }
```

Implementação da *activity* principal (tela de *login* do agente) – parte 4
Fonte: do autor

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_buscar_paciente);
    addCheckCPFCriptog();
    edtPaciente = (EditText)findViewById(R.id.editTextPaciente);
    edtPaciente.addTextChangedListener(new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence charSequence, int i, int i2, int i3) {

        }
        @Override
        //No evento de mudança de texto, sinaliza em preto o dado de identificação do paciente, caso
        //seja inferior a 10 caracteres, senão sinaliza em branco
        public void onTextChanged(CharSequence charSequence, int i, int i2, int i3) {
            if (edtPaciente.length() < 10){
                edtPaciente.setTextColor(Color.BLACK);
            }else{
                edtPaciente.setTextColor(Color.WHITE);
            }
            //Se o dado informado conter um quebra de linha (ultimo caracter lido pelo leitor
            //ou podendo ser um enter pressionado pelo agente), é removida a quebra de linha e
            //realizada a chamada ao metodo "buscarPaciente"
            if (edtPaciente.getText().toString().indexOf("\n") >= 0){
                edtPaciente.setText(edtPaciente.getText().toString().replace("\n",""));
                try {
                    buscarPaciente(findViewById(R.id.buttonBuscar));
                } catch (SQLException e) {
                    e.printStackTrace();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
```

Implementação da *activity* secundária (tela de busca do paciente) – parte 1
Fonte: do autor

```
private void addCheckCPFCriptog() {
    cbCript = (CheckBox)findViewById(R.id.cbCriptogCpf);
    cbCript.setOnClickListener(new OnClickListener() {

        @Override
        //Função do checkbox para associar alguns CPF's com sua criptografia
        //(apenas para demonstrar as buscas por CPF normal e CPF criptografado)
        public void onClick(View view) {
            if (cbCript.isChecked()){
                aplicaCPFCriptog(edtPaciente.getText().toString());
            }else{
                aplicaCPF(edtPaciente.getText().toString());
            }
        }
    });

}
```

Implementação da *activity* secundária (tela de busca do paciente) – parte 2
Fonte: do autor

```
    public static String resultDadosPac = "";
    //Função que determina o meio de identificação do paciente (UID Tag, CPF ou CPF criptog.)
    public void buscarPaciente (View view) throws SQLException, InterruptedException {
        Button b = (Button) view;
        String mensagem = "";
        //Sai da função caso o campo de busca do CPF esteja vazio
        if (edtPaciente.length() == 0) {
            return;
        }
        //Se conter até 10 caracteres, sabe-se que a identificação vem do código da tag
        if (edtPaciente.length() <= 10) {
            resultDadosPac = dadosPaciente(edtPaciente.getText().toString(), "", "");
        //Se conter 11 caracteres, a identificação trata-se do CPF do paciente
        }else if (edtPaciente.length() == 11){
            resultDadosPac = dadosPaciente("", edtPaciente.getText().toString(), "");
        //Por fim, se tiver mais de 11 caracteres, deduz-se que se trata do CPF criptog. do paciente
        }else if (edtPaciente.length() > 11){
            resultDadosPac = dadosPaciente("", "", edtPaciente.getText().toString());
        }
        //Se o texto da chamada da função acima for diferente de "paciente desconhecido", chama o
        //método que instancia a tela com os dados do paciente
        if (!resultDadosPac.equals("paciente desconhecido")) {
            TelaDadosPaciente(R.layout.activity_dados_paciente);
        }else{ //Caso o texto seja igual a "paciente desconhecido", mostra alerta de paciente não localizado
            mensagem = "Paciente Não Localizado";
            android.app.AlertDialog alertDialog;
            alertDialog = new AlertDialog.Builder(this).create();
            alertDialog.setTitle("Dados do Paciente");
            alertDialog.setMessage(mensagem);
            alertDialog.show();
        }
    }
```

Implementação da *activity* secundária (tela de busca do paciente) – parte 3
Fonte: do autor

```
    //Função que chama o método do webservice "buscaPaciente"
    public String dadosPaciente(String uid_tag, String cpf, String cpf_criptog){
        try {
            SoapObject buscaPaciente = new SoapObject(DadosMedicosActivity.NAMESPACE, DadosMedicosActivity.BUSCA_PACIENTE
            //Adicionando as propriedades "uid_tag", "cpf" e "cpf_criptog" ao objeto SOAP
            buscaPaciente.addProperty("uid_tag", uid_tag);
            buscaPaciente.addProperty("cpf", cpf);
            buscaPaciente.addProperty("cpf_criptog", cpf_criptog);

            SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11); //Criando o envelope
            envelope.setOutputSoapObject(buscaPaciente); //Adicionando o objeto SOAP ao envelope
            envelope.dotNet = true;
            HttpTransportSE http = new HttpTransportSE(DadosMedicosActivity.URL); //Criando objeto HTTP
            http.call(DadosMedicosActivity.SOAP_ACTION, envelope); //Efetuando a requisição
            //Capturando em bytes os dados do paciente
            byte array[] = envelope.getResponse().toString().getBytes("UTF-8");
            String resultString = new String(array, "UTF-8"); //Convertendo bytes em texto
            //Retorna o texto contendo os dados médicos do paciente
            return resultString;
        } catch (SoapFault soapFault) {
            soapFault.printStackTrace();
            return "erro soapFault";
        } catch (XmlPullParserException e) {
            e.printStackTrace();
            return "erro XmlPullParserException";
        } catch (HttpResponseException e) {
            e.printStackTrace();
            return "erro HttpResponseException";
        } catch (IOException e) {
            e.printStackTrace();
            return "erro IOException";
        }
    }
```

Implementação da *activity* secundária (tela de busca do paciente) – parte 4
Fonte: do autor

```
    //Instanciando a tela dos dados do paciente
    public void TelaDadosPaciente (int view){
        Intent it = new Intent(this, DadosPacienteActivity.class);
        startActivity(it);
    }
    //Ao retornar da tela dos dados do paciente para esta tela,
    //limpa o campo de busca do paciente e
    //desmarca o check de CPF criptog.
    public void onResume(){
        super.onResume();
        edtPaciente.setText("");
        cbCript.setChecked(false);
    }
```

Implementação da *activity* secundária (tela de busca do paciente) – parte 5
Fonte: do autor

```
public class DadosPacienteActivity extends Activity {

    private TextView tvDadosPac;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_dados_paciente);
        tvDadosPac = (TextView)findViewById(R.id.TextViewDadosPac);
        //Aplica ao componente de texto os dados do paciente
        tvDadosPac.setText(BuscarPacienteActivity.resultDadosPac);
    }
```

Implementação da *activity* terciária (tela com os dados médicos do paciente)
Fonte: do autor