INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

SISTEMA DE GESTÃO E DISPONIBILIZAÇÃO DE CONTEÚDOS ATRAVÉS DE NFC

Ricardo Jorge Gonçalves Portela

(Bacharel em Engenharia Informática e de Computadores)

Trabalho de Projecto para obtenção do grau de Mestre em Engenharia Informática e de Computadores

Orientador

Pedro Alexandre de Seia e Cunha Ribeiro Pereira

Co-Orientador

Paulo Alexandre Leal Barros Pereira

Júri

Fernando Manuel Gomes de Sousa

Jorge Manuel Rodrigues Martins Pião

Vítor Manuel Silva Rodrigues

Novembro de 2008

Resumo

A introdução de capacidades RFID (*Radio Frequency IDentification*) em dispositivos móveis não é recente. No entanto, a indústria nunca se mostrou muito interessada nesta vertente, porque não existia uma entidade que regulasse a normalização dos diversos aspectos relacionados, nomeadamente a forma como os dados são transferidos entre os dispositivos e de que forma são guardados esses dados nas *tags* RFID (dispositivos compostos por circuito integrado e antena, que podem armazenar dados e transferi-los por rádio frequência). A inexistência de normas que pudessem ser adoptadas pelos fabricantes levaria a que cada um tivesse que definir o seu próprio conjunto de formas de transferência e armazenamento, o que conduziria inevitavelmente à fragmentação do mercado. Estes problemas foram resolvidos com a criação da associação industrial sem fins lucrativos *NFC Forum*, que é composta por diversos fabricantes com interesse na área. Desde a sua criação, esta associação tem vindo a desenvolver protocolos com o objectivo de unificar dispositivos e soluções de vários fabricantes.

O termo NFC, ou *Near Field Communication*, foi criado para identificar esta nova geração de dispositivos móveis, que combinam as capacidades de processamento e interacção com o utilizador de um dispositivo móvel, com a comodidade associada à utilização de cartões sem contacto. Apesar de ser o elemento central nesta nova tecnologia, os dispositivos móveis não são os únicos que compõem esta nova família de tecnologias. Dela fazem também parte *tags* NFC, baseadas nos mesmos princípios das *tags* RFID, com a diferença que, nas primeiras, a forma como são organizados os dados é especificada pelas normas do *NFC Forum*.

Uma das áreas de aplicação da tecnologia NFC é a disponibilização de conteúdos presentes em *tags* ou em outros dispositivos NFC, e exibição desses conteúdos no dispositivo móvel.

No presente trabalho, são analisadas as normas publicadas pelo *NFC Forum* de forma a identificar a forma de estender a estrutura de mensagens proposta. A fase de análise incluiu também a identificação das características dos diversos dispositivos envolvidos nesta nova tecnologia (telemóveis, leitores desktop e *tags*). São apresentados os resultados do estudo de alguns projectos que utilizam tecnologia NFC, de forma a verificar até que ponto é interessante generalizar a plataforma a desenvolver neste projecto, bem como quais as funcionalidades mínimas desejáveis. Com base nesta análise, são definidos alguns conceitos base para a solução que envolvem tópicos como os protocolos de transporte, assim como o formato dos dados utilizado para transportar o conteúdo entre os vários componentes da solução.

O resultado é uma solução híbrida que admite um vasto leque de aplicações. Verificou-se que as soluções actuais poderiam ser implementadas com recurso à solução aqui apresentada e, nos casos em que tal não acontece directamente, a solução actual pode ser estendida para o tornar possível.

Palavras-chave

Near Field Communication, distribuição de conteúdos, *Bluetooth*, GPRS, *Mifare*, NFC-IP.

Abstract

The idea of incorporating Radio Frequency Identification (RFID) capabilities in a mobile device is not new. However, there was no real investment in this kind of products, mostly due to the fact that there was no regulatory entity and there were no open standards that could be used to guarantee that simple things, like content storage and exchange, were interoperable between the several existing manufacturers. These problems were solved with the creation of the non-profitable industrial association NFC Forum, which was founded by a group of companies interested in the development of this type of technologies. Since then, many have joined, and nowadays there are more than 150 members. Since its creation, this association has created standards that specify how the different components of the Near Field Communication (NFC) technology relate and communicate.

Near Field Communication (NFC) is the name of this new generation of mobile devices, which combine the processing and user interaction capabilities of a mobile device, with the commodity and ease of use associated with contactless cards. Even though the mobile phones are the main target and the source of innovation in this area, any other consumer electronics device can be made to use this new technology. NFC tags (devices that are composed of an integrated circuit and an antenna, capable of storing and transferring data over radio frequency modulation), have also a central role in this new technology. These tags can be specifically designed according to NFC Forum's tag type specifications, but can also be regular RFID tags. In this case, the content structure is regulated by NFC Forum's specifications.

One of the main application areas for this new technology is the delivery of content stored in *tags* or in other NFC devices, and the display of these contents in the mobile device.

In the present work, the NFC Forum's specifications are analyzed in order to identify how the proposed message organization is extendable. The analysis also covers other devices related to this new technology, such as desktop readers and tags.

To identify the typical applications of this type of platforms, an additional study is performed on the few existing projects that make use of NFC and that have been publicized in the last months.

This analysis enabled the creation of the concepts that support the presented solution, which involve topics such as transport protocols, as well as the data format used to transport the contents between the components that are part of the proposed solution.

The result is a hybrid solution that supports a wide set of application scenarios. This statement is supported by identifying how all the analyzed projects can be implemented using this solution.

Keywords

Near Field Communication, Content Delivery, *Bluetooth*, GPRS, *Mifare*, NFC-IP.

Agradecimentos

À minha família pelo apoio e pela estabilidade que me proporcionam.

Aos meus orientadores, Paulo Pereira e Pedro Pereira, por terem estado disponíveis nos momentos cruciais, e pelas excelentes revisões que efectuaram a este documento.

Ao Paulo Vieira, ao Ricardo Gregório e ao Valter Neves, não só pelas revisões que fizeram a este documento, mas por tudo o que têm feito nestes últimos anos de ISEL.

Aos Engenheiros António Calado e Pedro Silva do *Microsoft Language Development Center*, pela flexibilidade laboral que me concederam nos meses que antecederam a entrega deste documento.

Ao Engenheiro João Freitas, também do MLDC, pelos conselhos dados para a escrita deste documento.

Finalmente, gostaria de agradecer à equipa da Movensis pelo fantástico ambiente de trabalho e pela forma como me receberam, e em particular ao Professor Vítor Rodrigues, pelo convite de integrar o I&R e pela oportunidade de realizar um trabalho esta área.

Lista de abreviaturas

AMS	Application Management Software
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AWT	Abstract Window Toolkit
CA	Certificate Authority
CLDC	Connected Limited Device Configuration
CLI	Common Language Infrastructure
CDC	Connected Device Configuration
CSS	Cascading Style Sheet
DEP	Data Exchange Protocol
EA	Entidade Associação (Modelo)
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol
ISEL	Instituto Superior de Engenharia de Lisboa
IST	Instituto Superior Técnico
JAD	Java Application Descriptor
JAR	Java Archive
Java ME	Java Micro Edition
JCP	Java Community Process
JSON	Java Script Object Notation
JSR	Java Specification Request
LED	Light Emitting Diode
LINQ	Language INtegrated Query
MAC	Media Access Control (de Endereço)
MAC	Message Authentication Code (de Segurança)
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extensions
MMAPI	Mobile Media API
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NFC-IP	Near Field Communication – Interface and Protocol
ORM	Object-Relational Mapping
OTA	Over-The-Air
PAN	Personal Area Network
PDA	Personal Digital Assistant
RAR	Recommended Action Record
REST	REpresentational State Transfer

RFID	Radio Frequency Identification
RFU	Reserved for Future Use
RTD	Record Type Definition
SMS	Short Message Service
SQL	Structured Query Language
TAMA	Nome de código do micro controlador PN-531 do fabricante NXP Semiconductors
UART	Universal Asynchronous Receiver/Transmitter
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTF-8	8-bit Unicode Transformation Format
WIMA	Wireless Information Multimedia Applications (conferência)
WPF	Windows Presentation Foundation
XML	eXtensible Markup Language

Índice de Matérias

1 Introdução 1
1.1 Motivação 1
1.2 Objectivos 2
1.3 Contexto 3
1.4 Organização do documento 3
1.5 Contributos deste trabalho 3
2 Trabalho relacionado 5
2.1 Apresentação dos trabalhos 5
2.1.1 Cidade de Caen – Apresentação de monumentos e locais históricos 5
2.1.1.1 Apresentação da solução 5
2.1.1.2 Detalhes técnicos da solução 6
2.1.2 WIMA 2008 – Passeio pedonal Princesa Grace do Mónaco 7
2.1.2.1 Detalhes técnicos da solução 8
2.1.3 Hotspots Bluetooth 9
2.2 Análise dos trabalhos 9
2.2.1 Cidade de Caen – Apresentação de monumentos e locais históricos 9
2.2.2 WIMA 2008 – Passeio pedonal Princesa Grace do Mónaco 10
2.2.3 Hotspots Bluetooth 10
2.3 Conclusão 11
3 Modelo conceptual 13
3.1 Documentos e Elementos 13
3.2 Elementos de ligação 13
3.3 Servidores de conteúdos 15
3.3.1 Servidor de disponibilização de conteúdos via Bluetooth 15
3.3.2 Servidor de disponibilização de conteúdos via HTTP 16
3.4 Modelo de transferência de dados 16
3.4.1 Formato de seriação de dados 17
3.5 Sumário do capítulo 19
4 Arquitectura 21
4.1 Modelo de distribuição da aplicação de visualização de conteúdos 22
4.2 Cenários de utilização 24
4.3 Sumário do capítulo 28
5 Implementação 29
5.1 Servidor de dados 29
5.2 Camada de acesso a dados 30
5.3 Motor de Seriação de dados 34
5.3.1 Suporte adicional para o editor gráfico 36
5.4 Editor gráfico para criação / gestão dos conteúdos 37
5.4.1 Estrutura geral 38

5.4.2 Motor de exibição 40
5.4.3 Componentes de edição de documentos 41
5.4.3.1 Inserção de novos elementos de informação 41
5.4.3.2 Edição de propriedades dos elementos actuais 41
5.4.3.3 Edição de informação multimédia 44
5.4.3.4 Manipulação e organização dos elementos dentro do documento actual 44
5.4.4 Componentes de escrita ou envio de conteúdos 45
5.4.4.1 *Tags* Mifare 45
5.4.4.2 Ficheiro 46
5.4.5 Componentes para facilitação do processo de edição de conteúdos 46
5.4.5.1 NFC-IP 46
5.4.5.2 Bluetooth 46
5.5 Bibliotecas .NET para suporte à tecnologia NFC 46
5.5.1 Biblioteca de comunicação com leitores externos 46
5.5.1.1 Apresentação técnica do leitor NFC utilizado 47
5.5.1.2 Implementação da componente de acesso a *tags* Mifare 48
5.5.1.3 Interface de testes para a componente de escrita de *tags Mifare* 50
5.5.1.4 Implementação do suporte para o protocolo NFC-IP1 52
5.5.2 Implementação da especificação JSR-257 53
5.6 Aplicação *Java* ME para visualização dos conteúdos em dispositivos móveis 54
5.6.1 Plataforma *Java* ME 54
5.6.2 Apresentação técnica do protótipo utilizado 54
5.6.3 Arranque automático da aplicação 57
5.6.4 Assinatura digital da aplicação 58
5.6.5 Distribuição e instalação da aplicação 61
5.6.5.1 Distribuição e instalação baseada num sistema OTA 62
5.6.5.2 Instalação da aplicação de forma automática, utilizando mensagens NDEF 63
5.6.6 Criação de *Streams* para encapsular a escrita e leitura de dados através de NFC-IP1 64
5.6.7 Ambiente de desenvolvimento 65
5.6.7.1 *J2Me Polish* 66
5.6.7.1.1 Ambiente de compilação automática 66
5.6.7.1.2 Compilação condicional 67
5.6.7.1.3 Base de dados com informação acerca de dispositivos móveis 67
5.6.7.1.4 Seriação automática 68
5.6.7.1.5 Automatização de tarefas de rotina 68
5.6.7.1.6 Motor gráfico com suporte para definição de estilos 70
5.6.7.1.7 Ferramentas de localização da aplicação 71
5.6.7.1.8 Escolha condicional de ficheiros de suporte 72
5.6.7.1.9 Compilação simultânea para vários dispositivos 73
5.6.7.1.10 Funcionalidades Java5 74
5.6.8 Modelo de carregamento e apresentação de conteúdos 74

5.6.8.1 Descarregamento e apresentação de conteúdos multimédia 75
5.6.8.2 Utilização do motor de carregamento e exibição de conteúdos para outros fins 75
5.7 Web Service para acesso ao servidor de dados 77
5.8 Servidores de conteúdos 78
5.8.1 Bluetooth 78
5.8.2 HTTP 78
5.9 Sumário do capítulo 78
6 Resultados e Trabalho Futuro 81
6.1 Resultados 81
6.2 Trabalho futuro 82
7 Anexos 85
7.1 Apresentação dos quatro tipos de *tags* NFC 85
7.2 Modelo E-A da camada de persistência 86
8 Referências 87

Índice de Figuras

Figura 1 – Exemplo de painel informativo com capacidades NFC, em Caen - França 6
Figura 2 – Identificação dos diferentes tipos de *records* de uma mensagem NDEF 7
Figura 3 – Solução utilizada para a identificação de uma *tag* 8
Figura 4 – Exemplo de conteúdo obtido a partir da utilização do sistema 8
Figura 5 – Exemplo da utilização de um elemento de ligação 14
Figura 6 – Exemplo da utilização de elementos de ligação para a obtenção de conteúdo multimédia 14
Figura 7 – Formato de seriação dos conteúdos 17
Figura 8 – Formato do campo "Transformações" 17
Figura 9 – Formato de seriação de um documento 18
Figura 10 – Formato de seriação da dimensão 18
Figura 11 – Exemplo de utilização do formato de seriação da dimensão 18
Figura 12 – Formato de seriação de um Elemento 18
Figura 13 – Formato de seriação de uma propriedade 19
Figura 14 – Formato de seriação de uma cor 19
Figura 15 – Exemplo da implementação da solução proposta num museu 22
Figura 16 – Cenário de utilização – Todo o conteúdo cabe na *tag* 24
Figura 17 – Cenário de utilização – Parte do conteúdo na *tag*, restante obtido através de *Bluetooth* 25
Figura 18 – Cenário de utilização – Parte do conteúdo na *tag*, restante obtido através de HTTP 26
Figura 19 – Cenário de utilização – Transmissão de conteúdo através do protocolo NFC-IP (*Peer to Peer*) 27
Figura 20 – Cenário de utilização – Transmissão de conteúdo através do protocolo NFC-IP entre dois telemóveis 28
Figura 21 – Excerto do modelo E-A relacionado com o armazenamento dos conteúdos 30
Figura 22 – Comparação entre o modelo Entidade-Associação e o modelo de objectos criado pela ferramenta de ORM (caso N:M) 31
Figura 23 – Comparação entre o modelo Entidade-Associação e o modelo de objectos criado pela ferramenta de ORM (casos 1:N e 1:1) 33
Figura 24 – Diagrama do funcionamento da seriação dos conteúdos 34
Figura 25 – Diagrama do funcionamento da deseriação dos conteúdos 35
Figura 26 – Aspecto da aplicação de edição de conteúdos 38
Figura 27 – Estrutura geral da aplicação de edição de conteúdos 39
Figura 28 – Hierarquia (parcial) de controlos de exibição de elementos 40
Figura 29 – *Factory* criada para obter um controlo para um dado elemento 41
Figura 30 – Janela de inserção de novos elementos 41
Figura 31 – Apresentação do valor definido com a ajuda do editor de tipo 42
Figura 32 – Editor de tipo para identificadores de conteúdos remotos 42
Figura 33 – Editor de tipo para data e hora 43
Figura 34 – Utilização de conversores de tipo 43
Figura 35 – Editor de elementos de informação multimédia 44
Figura 36 – Componentes para manipulação e organização dos elementos 45

Figura 37 – Interface de gravação de *tags* Mifare 45
Figura 38 – Aspecto do leitor NFC utilizado 47
Figura 39 – Detalhe da estrutura do leitor Arygon APPA 47
Figura 40 – Interface que define o conjunto mínimo de operações que um leitor/escritor de *tags* Mifare tem de suportar 48
Figura 41 – Esquema das camadas de software e hardware relacionadas com o leitor ADRA (Arygon) 49
Figura 42 – Interface de testes para a componente de escrita de *tags Mifare* 51
Figura 43 – Diagrama de classes da aplicação de testes para a componente de escrita de *tags Mifare* 51
Figura 44 – Função de pré-visualização de conteúdos com envio através de NFC-IP1 52
Figura 45 – Diagrama de classes da implementação da especificação JSR-257 53
Figura 46 – Arquitectura do terminal utilizado para o desenvolvimento 54
Figura 47 – Aspecto do terminal utilizado, e indicação do local da antena NFC 55
Figura 48 – Relação entre os diferentes modos de funcionamento e restantes dispositivos NFC 56
Figura 49 – Processo de instalação típico utilizando um sistema de distribuição OTA 63
Figura 50 – Organização das mensagens NDEF que suportam a instalação automática da aplicação 64
Figura 51 – Aspecto do controlo *FileSystemStatus* 70
Figura 52 – Organização dos recursos de acordo com a cultura 71
Figura 53 – Estrutura de ficheiros de suporte à escolha condicional 73
Figura 54 – Classe *NFCInfoElement* 74
Figura 55 – Classe `NFCInfoControlBase` 75
Figura 56 – Menu de ajuda da aplicação 76
Figura 57 – Componentes que suportam a pré-visualização de conteúdos através de *Bluetooth* 76

Índice de Exemplos

Exemplo 1 – Obtenção de todos os projectos filtrados pelo seu nome, através do *DataContext* 32
Exemplo 2 – Utilização das classes geradas pela ferramenta de ORM (caso N:M) 32
Exemplo 3 – Utilização das classes geradas pela ferramenta de ORM (casos 1:N e 1:1) 33
Exemplo 4 – Interrogações através da utilização das funcionalidades da linguagem C# 3.0 34
Exemplo 5 – Excerto de um elemento e de uma propriedade desse elemento, anotados com atributos 35
Exemplo 6 – Utilização de atributos para associação de controlos gráficos a elementos de informação 40
Exemplo 7 – Associação de um editor de tipo a uma propriedade 43
Exemplo 8 – Ficheiro de configuração do modelo de providers de leitores *Mifare* 48
Exemplo 9 – Processamento de comando e utilização de `ConnectionProviders` 50
Exemplo 10 – Utilização dos serviços das camadas inferiores, na camada `LowLevelCommands`, retornando uma instância de um tipo composto 50
Exemplo 11 – Utilização dos serviços das camadas inferiores, na classe *ArygonAPPAReader* 50
Exemplo 12 – Utilização da classe *NFCIP1OutputStream* para o envio de conteúdos 52
Exemplo 13 – Excerto do ficheiro JAD, exibindo o atributo responsável pelo registo no *push registry* 58
Exemplo 14 – Conteúdo de um ficheiro JAD, correspondente a uma aplicação com assinatura digital 61
Exemplo 15 – Excerto de um ficheiro JAD, onde é identificada a localização do ficheiro JAR associado 62
Exemplo 16 – *Target* Ant 66
Exemplo 17 – Importação da *task* de compilação J2Me Polish 66
Exemplo 18 – Compilação condicional 67
Exemplo 19 – Utilização de variáveis obtidas a partir da base de dados de dispositivos 67
Exemplo 20 – Utilização da *framework* de seriação através do tipo *RmsStorage* 68
Exemplo 21 – Assinatura da *midlet* utilizando *J2Me Polish* 69
Exemplo 22 – Configuração do processo de ofuscação 69
Exemplo 23 – Configuração de uma ferramenta de compressão alternativa 69
Exemplo 24 – Atribuição de uma classe a um determinado controlo gráfico 70
Exemplo 25 – Excerto do ficheiro styles.css com a definição do estilo do controlo *FileSystemStatus* 71
Exemplo 26 – Excerto do ficheiro "messages.txt" com propriedades relacionadas com o controlo `FileSystemStatus` 72
Exemplo 27 – Exemplo de utilização da classe `de.enough.util.polish.Locale` para substituições simples 72
Exemplo 28 – Exemplo de utilização da classe `de.enough.util.polish.Locale` para substituições baseadas em argumentos 72
Exemplo 29 – Definição do conjunto de dispositivos para os quais a aplicação deve ser compilada 73
Exemplo 30 – Utilização de funcionalidades da plataforma *Java5* em *Java* ME 74
Exemplo 31 – Mapeamento entre pedidos REST [GET] e interrogações à base de dados 77
Exemplo 32 – Utilização de C#3.0 para efectuar interrogações ao serviço de dados 77

Índice de Tabelas

Tabela 1 – Tipos de dados suportados pelo *firmware* do dispositivo 56
Tabela 2 – Extensões e respectivos tipos MIME necessários num servidor de distribuição OTA 62

Capítulo 1

1 Introdução

Apesar de não ser uma tecnologia recente, a *Radio Frequency ID* (RFID), tem vindo a revelar-se uma tecnologia flexível e com vasto leque de aplicações no nosso quotidiano, desde cartões de identificação, de fidelidade e até cartões bancários. Estes últimos têm vindo lentamente a ser convertidos para esta tecnologia que apresenta vantagens em relação às soluções anteriores (geralmente baseadas em cartões de banda magnética, ou *SmartCards* com contacto), nomeadamente ao nível do desgaste, resistência aos elementos e rapidez de operação.

A mais recente aplicação desta tecnologia é a integração em dispositivos com capacidades de processamento e interacção com o utilizador. Devido à elevada penetração no mercado de dispositivos móveis, é nestes equipamentos que se espera maior evolução desta tecnologia.

A tecnologia *Near Field Communication* (NFC) apresenta como pontos fortes a simplicidade, a retro-compatibilidade com sistemas baseados em cartões sem contacto existentes e o valor acrescentado das capacidades que os dispositivos móveis trazem em termos de processamento, comunicação e interface com o utilizador. Outra das grandes motivações para a adopção desta tecnologia é o seu baixo custo. [1]

Para conduzir o processo de adopção e evolução desta tecnologia, bem como para garantir interoperabilidade entre os diversos fabricantes, foi criada a associação industrial sem fins lucrativos *NFC Forum*, onde participam cerca de 150 empresas que estão de alguma forma interessadas na evolução desta tecnologia. [2]

Uma das áreas de aplicação da tecnologia NFC, e foco deste trabalho, é a disponibilização de conteúdos, utilizando dispositivos com capacidades NFC.

1.1 Motivação

Durante os últimos 2 anos têm decorrido testes de implementação, suportados pelas mais diversas entidades. Um dos testes que mais relevo tem tido é o de Caen, uma cidade Francesa, onde a tecnologia NFC foi utilizada para facilitar acções do quotidiano, tais como a realização de pagamentos e obtenção de informações turísticas. [3] Actualmente, apesar das especificações *NFC Data Exchange Format* (NDEF) estarem concluídas, e preverem um conjunto de tipos de dados que podem ser trocados entre dispositivos NFC, apenas alguns tipos simples são suportados pelo *firmware* dos dispositivos NFC existentes. Tendo em conta esta limitação, a solução adoptada em Caen foi a escrita em *tags* NFC[1] da informação necessária para que na altura em que o dispositivo móvel toca na *tag*, seja efectuada uma chamada de voz para o número predefinido. Após o estabelecimento da chamada, é reproduzida uma gravação que apresenta o ponto de interesse onde o utilizador se encontra. Este é

[1] *Tag* NFC é o termo utilizado para referir qualquer *tag* de uma família de *tags* RFID que seja abrangida pelas especificações do NFC Forum em termos de organização da memória e da estrutura do seu conteúdo.

um dos tipos de registos actualmente suportados pelo *firmware* dos dispositivos NFC disponíveis. Apesar de esta ser uma utilização interessante do ponto de vista de acessibilidade, ficam por explorar os recursos que a maior parte dos dispositivos móveis actualmente dispõem, nomeadamente ecrã a cores de dimensões razoáveis e capacidades multimédia para áudio e vídeo. Esta solução não permite que o utilizador guarde o conteúdo para análise posterior, e existe o custo da chamada para a obtenção da informação que terá de ser suportado pela entidade que promove a divulgação da informação ou pelo próprio utilizador.

Outros tipos de interacção actualmente suportados incluem o arranque do *browser* do dispositivo móvel com o endereço carregado a partir de *tags* NFC, ou a obtenção de uma mensagem de texto previamente composta (corpo e destinatário), que poderá ser enviada com um simples clique.

Uma lacuna das soluções existentes é a entrega de conteúdo de dimensões superiores a 4KB[1]. Apesar das especificações do *NFC Forum* identificarem cenários de troca de informação directamente entre dispositivos NFC, através do protocolo *NFC-Interface Protocol Version 1* (NFC-IP1), estes cenários não são suportados em qualquer dos equipamentos disponíveis no mercado, que suportam apenas a troca de informação entre o dispositivo NFC e uma *tag* NFC. Estas *tags* têm capacidade de armazenamento reduzida, variando entre os 20 bytes e os 4KB. Esta limitação deixa de parte qualquer tentativa de distribuição de conteúdos como imagens, som e vídeo, de forma directa.

A possibilidade da entrega deste tipo de conteúdo permite novas aplicações para esta tecnologia, eliminando algumas das limitações existentes actualmente.

1.2 Objectivos

A limitação da capacidade das *tags* NFC é algo que não mudará num curto prazo de tempo, porque não existem planos dos fabricantes para o aumento da capacidade das mesmas. Com a tecnologia actual, o aumento da capacidade das *tags* iria contra um dos factores de sucesso desta tecnologia que é o seu baixo custo. Assim, a solução para a entrega de conteúdo de dimensões mais elevadas não pode passar apenas pelo armazenamento da informação em *tags*. Uma solução é a utilização de outros protocolos de comunicação para complementar esta lacuna.

Um dos casos de utilização da tecnologia NFC é o emparelhamento de dispositivos *Bluetooth* de forma automática através da utilização de *tags* NFC. Estas *tags* contêm a informação de emparelhamento que é posteriormente lida pelo dispositivo, desencadeando o processo. Um exemplo de aplicação deste caso de utilização é em impressoras: o utilizador do dispositivo móvel selecciona uma imagem, aproxima o terminal da *tag* associada à impressora, e a impressão é desencadeada de forma automática.

Partindo deste conceito, a solução pode passar pela obtenção de parte do conteúdo a partir da *tag* e do restante a partir de ligações de mais alto débito, aumentando o potencial conjunto de cenários de aplicação. A informação necessária para efectuar esta ligação é lida a partir da *tag*.

A solução deve ser flexível ao ponto de permitir a entrega de informação a partir de qualquer tipo de ligação existente no dispositivo móvel, bem como o armazenamento da informação para consulta posterior. Paralelamente, deverão existir formas intuitivas para a criação do conteúdo a entregar aos utilizadores.

[1] A dimensão é de cerca de 4KB, pois apesar de a capacidade total do modelo de *tag* em questão ser 4KB, parte deste espaço é utilizado em estruturas de controlo.

1.3 Contexto

A ideia para este projecto surgiu durante a realização de um estágio profissional por parte do aluno no departamento de inovação e pesquisa (I&R)[4] da empresa Movensis[5]. A sua implementação neste formato foi possível devido ao protocolo de colaboração existente entre esta empresa e o Instituto Superior de Engenharia de Lisboa (ISEL), e foi acompanhada na Movensis pelo Professor Vítor Rodrigues, director técnico do departamento I&R da Movensis e docente no Instituto Superior Técnico (IST).

1.4 Organização do documento

O documento actual encontra-se estruturado da seguinte forma:

- Capítulo 1, "Introdução" – Este é o capítulo actual que introduz o contexto de realização deste trabalho, o que motivou a sua realização e os objectivos que foram definidos.
- Capítulo 2, "Trabalho relacionado" – Neste capítulo são apresentados os projectos que representam o estado da arte na área abordada neste projecto.
- Capítulo 3, "Modelo conceptual" – Este capítulo introduz conceitos que foram desenvolvidos durante o desenho desta solução.
- Capítulo 4, "Arquitectura" – Neste capítulo é descrita a arquitectura desta solução, e são apresentados alguns casos de utilização para suportar a justificação das opções tomadas.
- Capítulo 5, "Implementação" – Este capítulo apresenta a forma como foi implementada a arquitectura proposta. Aqui é feita a descrição do *software* implementado, bem como do *hardware* necessário.
- Capítulo 6, "Resultados e trabalho futuro" – Neste capítulo é identificada a forma como os objectivos definidos inicialmente foram alcançados. São também definidas direcções para a evolução do trabalho actual.

1.5 Contributos deste trabalho

Este trabalho introduz um modelo extensível de transferência de informação que suporta a utilização de fontes alternativas de obtenção de conteúdos em complemento à leitura de *tags* NFC. Neste documento, para além da descrição do modelo e de como é extensível, são apresentadas implementações para o caso dos protocolos *Bluetooth* e HTTP. Este modelo, em conjunto com a arquitectura proposta permite a entrega de informação previamente definida, assim como a actualização da mesma em tempo real, com efeito imediato nos conteúdos apresentados aos utilizadores do sistema.

Para possibilitar a transferência de informação utilizando vários protocolos de transporte, foi definido o formato de seriação dos conteúdos. Este formato de seriação é extensível e permite especificar o conjunto de operações a serem aplicadas sobre o conteúdo, tais como compressão estatística ou utilização de esquemas de cifra simétrica.

É apresentada a implementação de uma aplicação que suporta a exibição de conteúdos em dispositivos móveis, utilizando o modelo de transferência de informação referido anteriormente. Para possibilitar a criação,

gestão e transferência dos conteúdos que são apresentados, é apresentada a implementação de uma aplicação de edição de conteúdos que permite a execução destas tarefas.

A escrita dos conteúdos desenhados na aplicação de edição de conteúdos em *tags* NFC é suportada pelo componente de escrita de *tags Mifare*. Uma vez que a informação escrita nas *tags* deve suportar não só o modelo de transferência de informação criado no âmbito deste trabalho, mas também as especificações do *NFC Forum*, foi criada uma biblioteca que permite a criação de mensagens de acordo com estas especificações, respeitando ambos os formatos.

Finalmente, para suportar cenários de geração dinâmica de conteúdos e respectiva transferência sem a necessidade da utilização de outras ligações sem fios, foi desenvolvida uma biblioteca que suporta o envio de informação utilizando o protocolo *peer-to-peer* NFC-IP1[6].

Capítulo 2

2 Trabalho relacionado

Este capítulo apresenta os projectos que representam o estado da arte na área da entrega de conteúdos através de NFC. Inicialmente os trabalhos são apresentados do ponto de vista funcional, e de seguida é feita uma descrição técnica dos mesmos. No final, são identificados os pontos fortes e fracos das soluções apresentadas.

2.1 Apresentação dos trabalhos

Sendo uma aplicação de uma tecnologia relativamente recente, existem poucos projectos com a mesma finalidade do trabalho apresentado. De seguida serão apresentados os projectos que se relacionam com o projecto actual.

2.1.1 Cidade de Caen – Apresentação de monumentos e locais históricos

O exemplo mais próximo do sistema a desenvolver é, pela forma de interacção com os utilizadores, o do teste de implementação realizado na cidade de Caen.[3] Este teste de implementação incluiu outros serviços tais como pagamentos e reservas de bilhetes que, por não se relacionarem directamente com este projecto, não serão aqui apresentados.

Uma das partes integrantes deste projecto é a apresentação de informação acerca de monumentos e pontos de interesse da cidade, com recurso à tecnologia NFC.

2.1.1.1 Apresentação da solução

Junto dos painéis informativos comuns, em papel, foi adicionada uma *tag* devidamente identificada, tal como é possível verificar na Figura 1. Esta permite aos detentores de dispositivos móveis com capacidades NFC a obtenção de informação acerca do monumento quer seja através da realização de chamada telefónica, ou do envio de mensagem via *Short Message Service* (SMS).

Figura 1 – Exemplo de painel informativo com capacidades NFC, em Caen - França

2.1.1.2 Detalhes técnicos da solução

Um dos tópicos que foi abordado pelo *NFC Forum* foi a forma como os dados são guardados nas *tags* NFC. Para este efeito, foi definido o formato para a troca de informação, denominado *NFC Data Exchange Format* (NDEF)[7].

As mensagens NDEF são compostas por *records* e, opcionalmente, cada *record* pode ainda conter *sub-records*. Cada *record* possui um tipo, e existe um conjunto predefinido de *records*, cada um deles desenhado para armazenar um tipo específico de dados.

Um exemplo de tipo de dados disponível em *records* predefinidos é o *Uniform Resource Identifier* (URI). Neste caso, o *record* obedece à especificação do tipo URI RTD (URI *Record Type Definition*)[8]. Este *record* é utilizado apenas para armazenamento de dados. No entanto, pode ser utilizado em conjunto com outro RTD[9], como por exemplo o *SmartPoster* RTD[10], que permite associar o URI a uma acção e ao *Text* RTD[11] que contém a descrição do URI. A acção indica o que deve ser feito pelo dispositivo que lê esta mensagem, nomeadamente:

1. Efectuar a acção
    a. Quando o URI identifica uma página Web, o dispositivo deve abrir o *browser* e navegar para o endereço especificado no *record* URI.
    b. Quando o URI identifica um número telefónico, deve ser estabelecida uma chamada.
    c. Quando o URI identifica uma mensagem de SMS, deve ser efectuado o envio da mesma.
2. Guardar para mais tarde
    a. Guardar o endereço nos favoritos, por exemplo
    b. Guardar o número na lista telefónica, com a descrição associada
    c. Guardar a mensagem de texto para envio posterior
3. Abrir para edição

A acção é definida através da utilização de um tipo local[1] ao *SmartPoster* RTD, denominado *Recommended Action Record* (RAR).

Para além dos tipos descritos anteriormente, existe ainda a possibilidade da criação de registos com tipos *Multipurpose Internet Mail Extensions* (MIME). A Figura 2 resume os tipos de *record* descritos anteriormente.

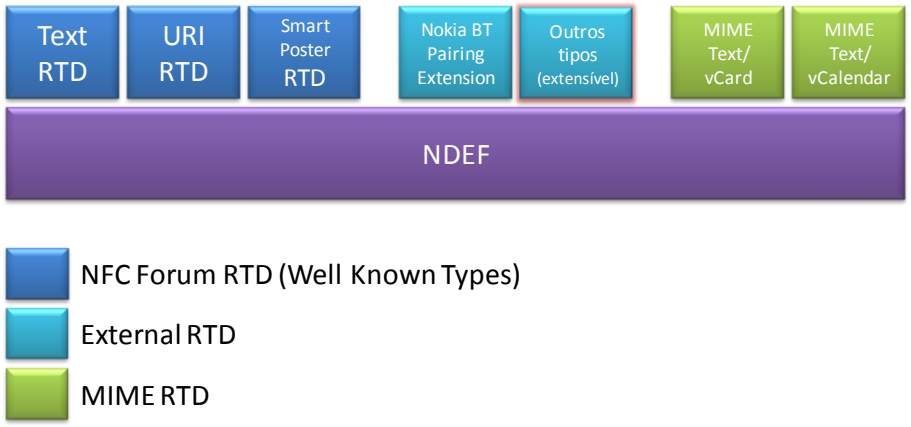


Figura 2 – Identificação dos diferentes tipos de *records* de uma mensagem NDEF

Este modelo é extensível através da definição de tipos externos, denominados *External RTDs*. Estes tipos são utilizados quando o conteúdo do *record* não se encaixa em nenhum dos tipos previamente descritos.
A solução implementada em Caen foi baseada em mensagens NDEF, com RTDs dos tipos *Text*, URI e *Smart Poster*, combinados para produzir os comportamentos descritos anteriormente.

Este funcionamento não requer a instalação de *software* adicional nos terminais dos utilizadores, porque utiliza tipos de dados que o *firmware* do dispositivo sabe interpretar e exibir. Isto sucede para alguns tipos, normalmente referidos por *well known types*.

2.1.2 WIMA 2008 – Passeio pedonal Princesa Grace do Mónaco

Durante a conferência WIMA 2008, foi apresentada uma solução que se encontra permanentemente disponível no Percurso Princesa Grace do Mónaco, um itinerário criado em honra desta princesa em 2007. [12] Este percurso consiste em 25 "etapas" que comemoram as acções mais marcantes da sua vida.

Para dar início à utilização desta solução, os utilizadores aproximam o dispositivo móvel de uma etiqueta devidamente identificada num dos placares existentes em cada etapa (Figura 3), e obtêm informação adicional acerca do local/evento (Figura 4).

[1] Um tipo local é um tipo de *record* que só faz sentido em determinado contexto. Neste caso, o tipo identificado só faz sentido quando utilizado em conjunto com um *record* do tipo *Smart Poster*.

Figura 3 – Solução utilizada para a identificação de uma *tag*

Figura 4 – Exemplo de conteúdo obtido a partir da utilização do sistema

Esta solução já apresenta algumas semelhanças com a solução pretendida. No entanto, é importante referir que esta solução é posterior à ideia que deu origem ao arranque do projecto apresentado neste documento. Não foi possível descobrir se existe a possibilidade de obter sons e vídeo. Também não foi possível apurar a capacidade de armazenamento da informação para análise posterior.

2.1.2.1 Detalhes técnicos da solução

Nesta solução é utilizada uma *midlet Java Micro Edition* (*Java* ME) que obtém dados a partir das *tags* que se encontram espalhadas pelo passeio pedonal. Esses dados são utilizados para a invocação de um *Web service*, utilizando a ligação *General Packet Radio Service* (GPRS) disponível no dispositivo móvel. Os dados obtidos através desta invocação são posteriormente apresentados no ecrã do terminal, na forma de um documento com imagens e texto formatado (cor, tipo de letra, entre outros).

Através da visualização de todas as demonstrações disponíveis desta solução, não foi possível visualizar o acesso a conteúdo gravado anteriormente no dispositivo. De igual modo, nunca foi exibido conteúdo multimédia tal como sons, vídeo e imagens de dimensão superior a um ícone/logótipo (na ordem dos 2000 pixéis). Esta restrição é normal tendo em conta a baixa largura de banda disponibilizada através de uma ligação de GPRS, e com o facto dos custos das comunicações com o serviço serem suportadas pelo cliente.

Relativamente às mensagens NDEF que foram armazenadas nas *tags* (NXP *Mifare* 1k [13]), para cumprir com as especificações do *NFC Forum*, a solução adequada seria a utilização de *External RTDs*. Não foi possível apurar tal facto, devido à existência de pouca informação técnica relativa à solução.

2.1.3 Hotspots Bluetooth

Em termos das funcionalidades e potencialidades multimédia, as soluções de envio de conteúdo utilizando o protocolo *Bluetooth*, conhecidas como *hotspots Bluetooth*, apresentam alguns conceitos relevantes para o projecto em questão. Nestes produtos são utilizadas as capacidades de procura de dispositivos disponíveis através de ligação *Bluetooth*, e é efectuado o envio de objectos para os dispositivos que sejam encontrados ao alcance do dispositivo de *broadcast*. Existem plataformas flexíveis que permitem a identificação das capacidades do dispositivo móvel (dimensão de ecrã e qual o suporte multimédia, por exemplo), optimizando o conteúdo que será apresentado no dispositivo. Isto garante que o conteúdo estará no formato mais indicado para o terminal em questão.

2.2 Análise dos trabalhos

Nesta secção, é feita a comparação das soluções existentes, identificando os pontos fortes e fracos de cada uma delas, bem como falhas comuns. Este levantamento é utilizado para influenciar as decisões tomadas na implementação da solução proposta.

2.2.1 Cidade de Caen – Apresentação de monumentos e locais históricos

Pontos fortes

- Acessibilidade e compatibilidade elevada, uma vez que se recorrem apenas aos tipos RTD suportados directamente pelo *firmware* dos dispositivos.
- Solução com potencial para pessoas que se encontram menos familiarizadas com a tecnologia, uma vez que não é necessário interagir com uma aplicação: basta fazer uma chamada de voz ou enviar um SMS.

Pontos fracos

- Não tira partido das capacidades de comunicação *Bluetooth* e GPRS, nem das capacidades multimédia e de interacção com o utilizador disponibilizados pelos dispositivos móveis.
- Existem custos associados às chamadas de voz/mensagens SMS que suportam o funcionamento do serviço.
- No caso das chamadas de voz, não é possível guardar o conteúdo para análise posterior.

2.2.2 WIMA 2008 – Passeio pedonal Princesa Grace do Mónaco

Pontos fortes

- Suporta a apresentação de imagens e texto formatado.
- O facto de se basear em conteúdos que se encontram num servidor central, facilita a actualização dos conteúdos, que até se podem tornar dinâmicos, por exemplo, para ir sabendo o desenrolar de uma partida de futebol, através de evocações periódicas do serviço.

Pontos fracos

- Custo das comunicações suportado pelo cliente.
- A utilização de GPRS limita o tipo de conteúdos que podem ser entregues. Apesar de na teoria poder ser enviado qualquer tipo de conteúdo, na prática não podem ser utilizados conteúdos de dimensão elevada para manter o serviço utilizável do ponto de vista de custos e de tempos de carregamento.
- Não foi possível apurar se é suportada a entrega de vídeos e som, mas tudo indica que não será possível, até porque tal está limitado à partida pela utilização de GPRS como meio de obtenção dos dados.

2.2.3 Hotspots Bluetooth

Pontos fortes

- Entrega optimizada de conteúdo, através da identificação dos terminais e das respectivas capacidades de exibição multimédia.
- A entrega de conteúdos é grátis uma vez que a utilização de ligações *Bluetooth* não acarreta custos adicionais.
- Entrega de conteúdo com taxas de transferência elevadas, quando comparadas com as obtidas usando GPRS.
- Compatibilidade com todos os terminais que suportem a recepção de objectos via *Bluetooth*.
- Não sendo ubíqua, é uma forma de ligação que ainda assim pode cobrir vastas áreas, com recurso a vários pontos de acesso.

Pontos fracos

- Do ponto de vista do protocolo *Bluetooth*, é necessário que os terminais do cliente estejam visíveis a todos os outros dispositivos. Esta exigência levanta problemas de segurança, uma vez que o utilizador poderá receber ficheiros nocivos enviados por utilizadores mal intencionados.
- O processo de preparação do envio pode demorar algum tempo, principalmente quando existem muitos dispositivos *Bluetooth* na zona.
- Implica a configuração do dispositivo por parte do utilizador. É necessário que o terminal tenha a ligação *Bluetooth* activa, e que esteja configurado para ser descoberto por outros dispositivos.

2.3 Conclusão

Após a análise das tecnologias existentes (e de certa forma, concorrentes), relacionadas com o presente trabalho, foram identificadas algumas características que serão tidas em consideração no desenho da solução final:

- A solução de Caen é muito interessante do ponto de vista de acessibilidade, pelo que deve ser possível obter o conteúdo sem forçar o utilizador a abrir uma aplicação no dispositivo móvel. O simples facto de encostar o aparelho numa *tag* devidamente identificada, deve ser o suficiente para desencadear a obtenção do conteúdo.
- A entrega de conteúdos de dimensões elevadas é um ponto a favor dos *hotspots Bluetooth*, pelo que deve ser suportada a obtenção de conteúdos por este meio. No entanto, deve ser evitada a necessidade de o utilizador manter o seu dispositivo visível para os outros terminais.
- Um dos pontos fortes da tecnologia NFC é o facto de esta ser uma tecnologia muito intuitiva, em que não é necessário proceder à configuração prévia dos dispositivos. Assim sendo, deve ser identificada uma solução simples para a instalação da *midlet*, sem ter de se recorrer à navegação a um sítio, ou ao descarregamento da aplicação via *Bluetooth* ou cabo de dados.
- Apesar de acarretar custos, deve ser mantida a opção de obtenção de conteúdos a partir de ligação GPRS. Isto permite que sejam suportados cenários de dispersão geográfica elevada. Para além disso, a tendência é para que no futuro este tipo de comunicações se torne mais económico.
- Deve ser possível armazenar localmente os conteúdos obtidos, permitindo assim que os utilizadores revejam mais tarde a informação obtida. Um cenário de utilização pode ser o suporte de campanhas de fidelização, com o armazenamento de *coupons* por esta via. Neste cenário cada *coupon* teria um identificador associado, gerado na altura do seu envio, e seria invalidado no sistema central após a utilização. A utilização do *coupon* realizar-se-ia através da sua exibição no dispositivo móvel, ou envio através de NFC-IP.
- Os conteúdos distribuídos devem suportar as capacidades avançadas que se podem encontrar nos dispositivos móveis actuais, nomeadamente:
    - Apresentação de texto formatado (com cores e fontes configuráveis)
    - Reprodução de áudio e vídeo
    - Navegação directa para *sites* Web
    - Realização automática de chamadas telefónicas
    - Envio de mensagens de mensagens de SMS pré formatadas
- A troca de informação directamente entre utilizadores deve ser possível. Isto possibilita a troca de informação previamente obtida.

Capítulo 3

3 Modelo conceptual

Neste capítulo serão apresentados os conceitos que dão suporte ao cumprimento dos requisitos identificados após a análise dos projectos existentes que se relacionam com este trabalho, apresentada no Capítulo 2.

3.1 Documentos e Elementos

Na solução desenvolvida, a criação de conteúdos tem como base o conceito de elemento. O elemento é a unidade mínima de informação que determinado documento pode conter. Cada elemento pode representar uma imagem, uma chamada de voz, um vídeo, uma mensagem de SMS, ou outro tipo de conteúdo. O documento pode incluir um número arbitrário de elementos.

3.2 Elementos de ligação

Para suportar a distribuição de conteúdos a partir de qualquer tipo de ligação de dados, foi criado o conceito de elemento de ligação. O elemento de ligação contém informação acerca de como obter determinado conteúdo, utilizando o tipo de ligação associado. A informação contida no elemento de ligação é tipicamente um endereço ou algo semelhante, dependendo do protocolo que serve de base ao elemento em questão. Esta informação é utilizada para aceder ao servidor de conteúdos associado. O conceito de servidor de conteúdos será apresentado na Secção 3.3. O servidor de conteúdos pode ser visto neste modelo como o conceito de componente servidora, sendo os elementos de ligação a componente equivalente, para a parte de cliente. Actualmente estão implementados elementos de ligação para os tipos de ligação *Hyper-Text Transfer Protocol* (HTTP)[14] e *Bluetooth*.

A Figura 5 apresenta um exemplo de utilização do elemento de ligação *Bluetooth*. Quando o utilizador selecciona o elemento de ligação é estabelecida a ligação *Bluetooth* ao servidor de conteúdos, utilizando os parâmetros armazenados no elemento. Após o estabelecimento da ligação, o conteúdo associado é obtido e exibido no ecrã do dispositivo. O conteúdo exibido encontra-se armazenado na forma de outro documento, criando um funcionamento semelhante ao das hiper-ligações em documentos de *Hyper-Text Markup Language* (HTML).

Outra utilização possível para os elementos de ligação é o funcionamento como fonte de dados para elementos multimédia. Desta forma podem ser associados, no mesmo documento, elementos de áudio a elementos de ligação HTTP, e elementos de vídeo a elementos de ligação *Bluetooth*, possibilitando que o áudio seja carregado através da ligação à Internet disponível no dispositivo, e que o vídeo seja carregado utilizando a ligação *Bluetooth*.

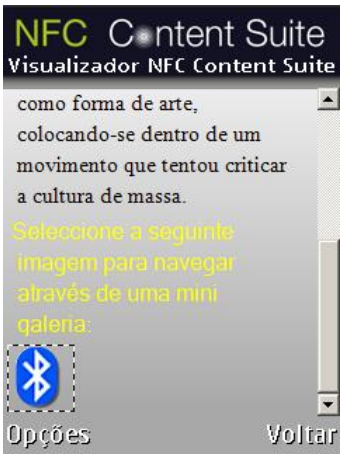


Figura 5 – Exemplo da utilização de um elemento de ligação

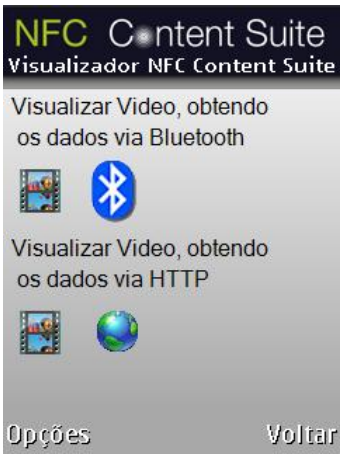


Figura 6 – Exemplo da utilização de elementos de ligação para a obtenção de conteúdo multimédia

A Figura 6 apresenta um exemplo de utilização de dois elementos de ligação, *Bluetooth* e HTTP, para a obtenção de conteúdos multimédia. Quando o utilizador selecciona o elemento de vídeo, é estabelecida a ligação *Bluetooth* ou HTTP (dependendo do ícone seleccionado) e é apresentada a interface de visualização de vídeos, onde o vídeo em questão será exibido. Tal como no caso anterior, a ligação é estabelecida utilizando os parâmetros armazenados no elemento de ligação associado. Esta utilização permite a selecção de ligação *Bluetooth* para obter o conteúdo quando está dentro do alcance do servidor de conteúdos *Bluetooth*, ou HTTP quando a ligação anterior não está disponível. Neste caso os elementos de ligação estão visíveis, mas é possível que não estejam. A visibilidade dos elementos é uma das suas propriedades e como tal, é configurável.

Esta abordagem permite que conteúdos complexos sejam obtidos a partir de fontes externas, sendo a informação inicial obtida a partir de uma *tag* de capacidade reduzida (que guarda apenas a informação necessária para estabelecer a ligação à fonte de dados).

Actualmente existem dois tipos de servidor de conteúdo: *Bluetooth* e HTTP. O servidor HTTP possibilita a entrega de conteúdo através de qualquer ligação que suporte o protocolo TCP/IP, nomeadamente, GPRS e *Wireless Ethernet*.

3.3 Servidores de conteúdos

De forma a suportar a obtenção de informação através de vários tipos de ligações, foi criado o conceito de servidor de conteúdos. O servidor de conteúdos e o elemento de ligação complementam-se: o elemento de ligação é utilizado para efectuar pedidos ao servidor de conteúdos, que se encarrega de os atender.

São fornecidas duas implementações de servidores de conteúdos: HTTP e *Bluetooth* (e dos respectivos elementos de ligação). Este modelo é extensível e pouco restrito. Os requisitos para servidores de conteúdos são os seguintes:

1. Suporte para a recepção de informação (argumentos tais como o identificador do conteúdo).
2. Suporte para o envio de dados.

Um exemplo de extensibilidade para este modelo seria a utilização de mensagens binárias de SMS para a obtenção de conteúdos. Para isto, seriam necessários os seguintes componentes:

1. Novo tipo de elemento de ligação que enviaria uma mensagem de SMS com os dados do pedido para um número conhecido.
    a. O formato da mensagem bem como o número de destino seriam propriedades de configuração do elemento.
2. Servidor de conteúdos que receberia mensagens binárias que iriam conter os dados para o pedido.
    a. Com base nos argumentos recebidos, o servidor consultaria o serviço de acesso a dados e enviaria o conteúdo dividido em tantas mensagens quantas as necessárias para o conteúdo em questão. Na implementação do cliente teriam de ser obtidas as mensagens vindas do número em causa, e tratadas para poderem ser utilizadas como um *stream*, a ser utilizado pelo motor de seriação implementado na aplicação *Java ME* de visualização de conteúdos, que será apresentada na Secção 5.6. Um *stream* encapsula uma sequência de *bytes*, permitindo acções tais como a leitura de parte ou da totalidade dessa informação. Os utilizadores deste tipo de objectos não necessitam de conhecer a forma como a sequência de *bytes* é obtida, permitindo a criação de componentes genéricos, tal como será apresentado na Secção 5.6.

Neste modelo, caso ocorram erros no servidor de conteúdos, é gerado de forma dinâmica um documento que obedece ao formato estabelecido, com a mensagem de erro apropriada. Desta forma, o utilizador obtém imediatamente informação acerca do que provocou a falha na obtenção do conteúdo (conteúdo expirado, não encontrado, etc.).

3.3.1 Servidor de disponibilização de conteúdos via Bluetooth

O servidor de disponibilização de conteúdos *Bluetooth*, bem como o elemento de ligação associado, suporta dois modos de funcionamento distintos. No primeiro, o elemento de ligação contém o endereço *Media Access Control* (MAC) do servidor *Bluetooth*. No segundo, o endereço MAC não é especificado, e o servidor é procurado através do suporte do protocolo *Bluetooth* para a procura de serviços em dispositivos.

Ambos os modos têm os seus pontos fortes e fracos:

- Utilização do endereço MAC
    - Pontos Fortes
        - Acesso imediato, desde que o servidor esteja disponível.
        - Não exige que o servidor de conteúdos esteja visível para todos os dispositivos.
    - Pontos Fracos
        - Não suporta a leitura da *tag* numa localização e consulta do restante conteúdo em movimento, porque se o utilizador sair do alcance do servidor, cujo endereço MAC está gravado na *tag* consultada, deixa de ser possível obter o conteúdo.
- Utilização da funcionalidade de pesquisa de serviços em dispositivos
    - Pontos Fortes
        - É possível suportar cenários em que o utilizador lê o conteúdo num local, desloca-se e obtém o restante conteúdo noutra sala, desde que a nova sala esteja também ao alcance de um servidor *Bluetooth.*
        - É suportado o balanceamento de carga por diversos servidores.
    - Pontos Fracos
        - Uma vez que é necessário confirmar o suporte do serviço por parte dos dispositivos que se encontram visíveis se determinado serviço é suportado, no pior dos casos, o servidor de conteúdos pode ser o último a ser questionado. Se existirem muitos dispositivos *Bluetooth* visíveis na área em questão, o passo de procura pode demorar muito tempo.

3.3.2 Servidor de disponibilização de conteúdos via HTTP

Este servidor é responsável por servir conteúdos via HTTP e cria a possibilidade da entrega de conteúdo a qualquer dispositivo que suporte este protocolo. Uma vez que existem várias formas de aceder a serviços HTTP, nomeadamente via *Wireless LAN, GPRS* e *Bluetooth*, a existência deste servidor aumenta de forma significativa o conjunto de dispositivos suportados pela aplicação.

3.4 Modelo de transferência de dados

Tendo sido identificada a necessidade da informação poder ser enviada por vários meios, bem como ser guardada em diferentes tipos de armazenamento, foi especificado o formato de seriação que transforma os elementos (objectos em memória) numa sequência de *bytes*. Este formato será utilizado para a escrita de conteúdo em *tags* NFC, bem como para o envio de conteúdo através de ligações NFC-IP1, *Bluetooth* ou HTTP. O armazenamento de conteúdos no sistema de ficheiros do dispositivo também é suportado através da seriação da informação utilizando este formato.

3.4.1 Formato de seriação de dados

Todos os conteúdos que são manipulados estão organizados dentro de um documento. Os documentos são seriados obedecendo ao formato apresentado na Figura 7.

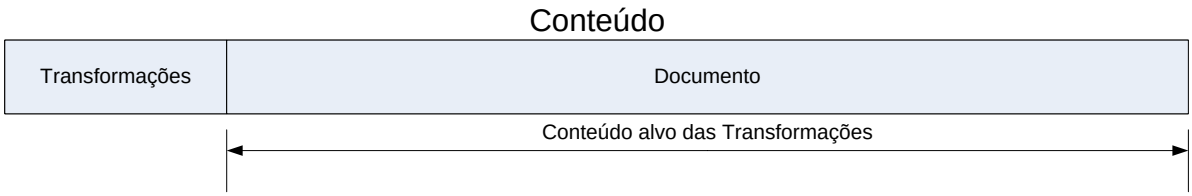


Figura 7 – Formato de seriação dos conteúdos

O campo "Transformações", detalhado na Figura 8, representa a forma como o conteúdo é codificado. Actualmente é utilizado para indicar a necessidade de usar um descompressor GZIP para obter o documento associado. Apesar de não terem sido implementadas outras transformações, utilizações possíveis para este campo incluem a sinalização da utilização de esquemas de cifra ou de *Message Authentication Code* (MAC) simétricos sobre o documento. A utilização de esquemas de cifra simétrica permitiria a transferência dos conteúdos de forma segura entre os vários componentes da solução, evitando que o conteúdo desses documentos fosse interpretado por terceiros. Para garantir a integridade dos dados presentes no documento, poderia ser utilizado um esquema MAC. No primeiro caso seria necessário estabelecer a forma de distribuição das chaves privadas aos clientes autorizados a decifrar os conteúdos. Para o segundo seria necessária a distribuição da chave pública de validação da marca do esquema MAC simétrico. Para os cenários de aplicação pensados para este projecto, estas funcionalidades não são necessárias. No entanto, o modelo é extensível e permite o seu suporte.

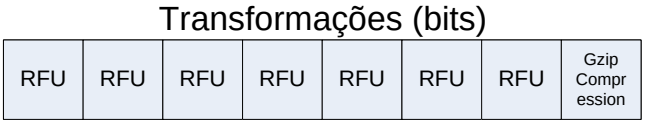


Figura 8 – Formato do campo "Transformações"

Actualmente, está implementada apenas a transformação *GZip Compression*. Os restantes bits neste campo estão reservados para uso futuro ("*RFU*").

Uma transformação é efectuada sobre todo o conteúdo do documento, e tem uma transformação inversa que repõe o conteúdo para o seu estado inicial. Este conjunto de operações pode ser visto como várias transformações que são efectuadas ao conteúdo original.

Na descodificação do conteúdo, quando são aplicadas todas as transformações inversas, é obtido um documento que segue a organização apresentada na Figura 9.

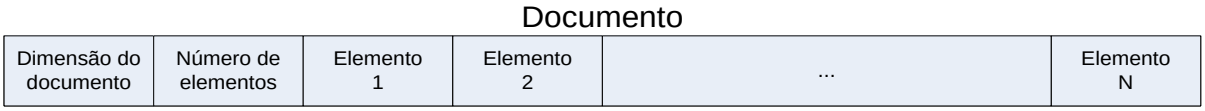


Figura 9 – Formato de seriação de um documento

O documento contém informação acerca da sua dimensão (em bytes) e do número de elementos pelos quais é composto.

Para representar as dimensões foi utilizada a estrutura apresentada na Figura 10.

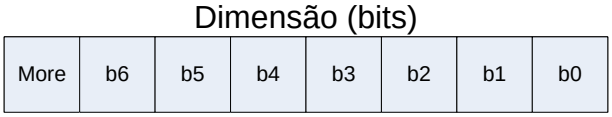


Figura 10 – Formato de seriação da dimensão

Como este campo será utilizado em todas as propriedades, é impossível escolher à partida qual o número de bytes que são necessários para especificar uma dimensão. Assim, a opção recaiu sobre o esquema dinâmico, apresentado de seguida. Neste esquema, se o bit de maior peso for "1" indica que o próximo byte deve ser concatenado e interpretado como uma extensão aos restantes 7 bits actuais.

Esta estrutura é recursiva, pelo que pode ser representada qualquer dimensão. Na Figura 11 apresenta-se um exemplo para o valor binário "11111111111111":

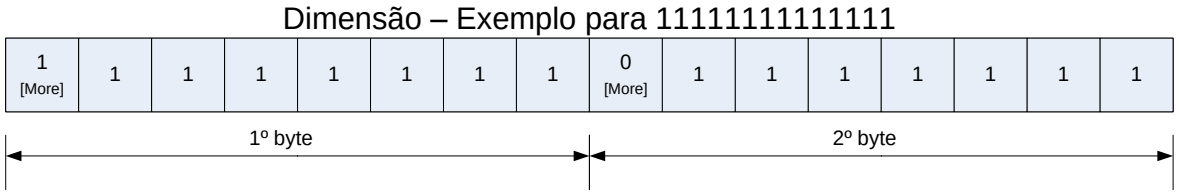


Figura 11 – Exemplo de utilização do formato de seriação da dimensão

Cada elemento contém o respectivo identificador (que é também utilizado para identificar quais as propriedades que podem surgir no elemento actual), e o número de propriedades que contém. Este ponto merece algum destaque. O tipo de elemento especifica o conjunto de propriedades que o compõem. Essas mesmas propriedades possuem valores por omissão. Por motivos de optimização da utilização de largura de banda e espaço de armazenamento, se na altura da seriação da propriedade esta possuir o valor por omissão, a propriedade não é seriada. Este aspecto justifica a existência do campo "Número de propriedades" na estrutura do Elemento, apresentada na Figura 12.

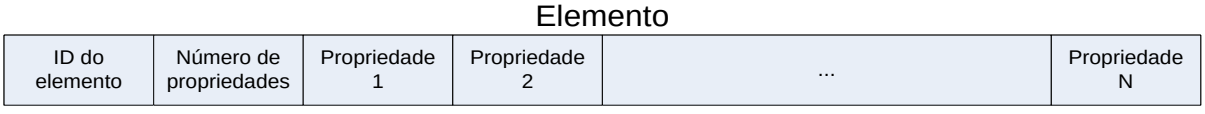


Figura 12 – Formato de seriação de um Elemento

Cada propriedade é composta pelo seu identificador e pelos dados associados à propriedade. Esta estrutura é apresentada na Figura 13. A forma como os dados da propriedade são codificados é identificada pelo campo "ID da propriedade". A necessidade da existência do identificador resulta da optimização identificada anteriormente, através da qual é apenas efectuado o envio das propriedades cujo valor é diferente do valor definido por omissão.

Figura 13 – Formato de seriação de uma propriedade

A codificação de cada propriedade, tal como foi dito, varia consoante o seu identificador. Na Figura 14 é apresentado um exemplo de tipo de dados e a respectiva codificação. Este é o tipo de dados utilizado para a representação de cores.

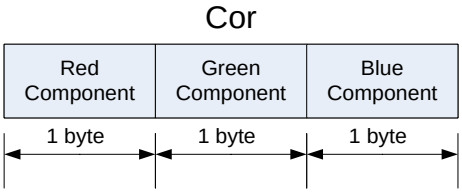


Figura 14 – Formato de seriação de uma cor

3.5 Sumário do capítulo

Neste capítulo foi identificada a forma como se encontra estruturada a informação que será manipulada neste projecto.

O documento foi identificado como a unidade mínima de entrega de informação. Este pode conter zero ou vários elementos. Estes elementos possuem propriedades que definem o seu aspecto e comportamento. O conjunto de propriedades existentes em determinado elemento varia consoante o seu tipo.

Foram também apresentados os conceitos que suportam a entrega de conteúdos de dimensões elevadas. Para isso foram criados os conceitos de elemento de ligação e servidor de conteúdos. O elemento de ligação é uma classe de elementos que permite obter informação de outras fontes, utilizando protocolos disponíveis no dispositivo móvel. A obtenção de informação é feita através da realização de pedidos a servidores de conteúdos.

Para a obtenção de informação através de outros protocolos de transferência de informação, foi especificado o formato de seriação que é utilizado para transformar os documentos e respectivos elementos em sequências de *bytes*.

Capítulo 4

4 Arquitectura

Neste capítulo é apresentada a forma como as diferentes partes que compõem a solução interagem, e qual o papel que desempenham. Seguidamente, são também apresentados cenários de aplicação dos componentes apresentados.

A solução proposta inclui um editor gráfico de conteúdos, a ser usado pelos gestores. Os gestores são os utilizadores da solução que desenham os conteúdos a ser exibidos. Esses conteúdos são armazenados em base de dados utilizando outro componente da solução, a camada de acesso a dados.

A entrega dos conteúdos pode ser efectuada através da utilização de um servidor de conteúdos ou de *tags* NFC. A escrita dessas *tags* é também efectuada na aplicação de edição de conteúdos. Para a exibição dos conteúdos aos utilizadores do sistema, foi criada uma aplicação *Java ME*. Para além da exibição de conteúdos lidos a partir de *tags* NFC, esta aplicação permite a obtenção de conteúdos de dimensão elevada a partir dos servidores de conteúdos *Bluetooth* e HTTP.

De forma a possibilitar o acesso aos conteúdos por parte dos servidores de conteúdos de forma remota, evitando expor o servidor de base de dados para a Internet, foi criado um *Web service* que age como intermediário no acesso ao serviço de dados.

A Figura 15 ilustra uma das aplicações possíveis da solução e apresenta o enquadramento de alguns dos componentes referidos anteriormente. Neste exemplo de implementação da solução num museu, os utilizadores podem obter conteúdos utilizando várias tecnologias. Em cada obra de arte, existem *tags* NFC com conteúdo relativo à obra, ao autor ou ao movimento artístico. As *tags* contêm a descrição resumida da obra em questão. Se o utilizador desejar, pode obter conteúdo mais detalhado através da utilização da ligação *Bluetooth* ou HTTP do dispositivo. Junto à bilheteira do museu, existe um computador que executa uma aplicação em modo quiosque que permite aos utilizadores a escolha das obras expostas e exibe informação associada no ecrã. Se o utilizador desejar, esse conteúdo pode ser transferido para o seu terminal NFC através da utilização do protocolo NFC-IP1.

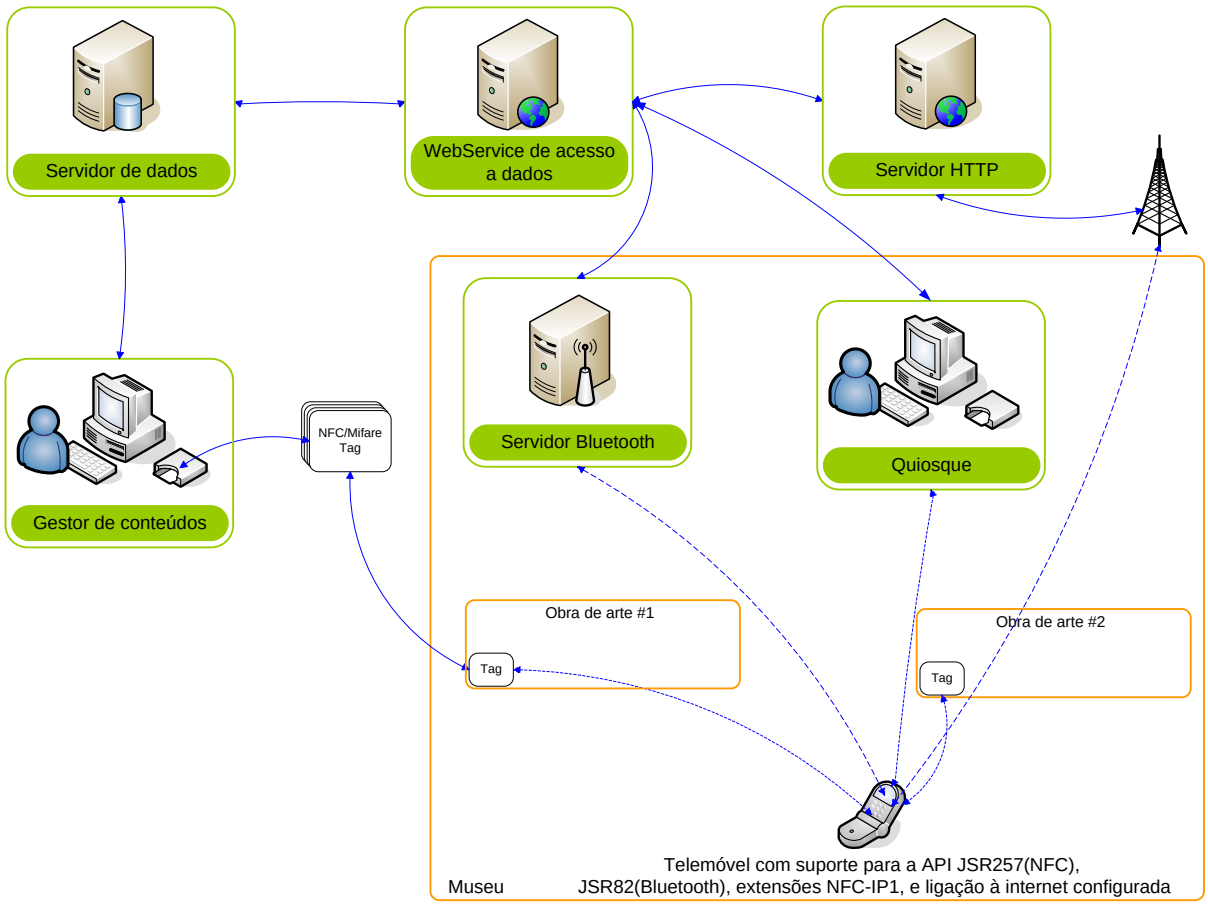


Figura 15 – Exemplo da implementação da solução proposta num museu

Nas obras de arte, os conteúdos são obtidos com recurso à utilização de *tags* NFC, em conjunto com outra tecnologia sem fios, quando necessário. A leitura de *tags* colocadas junto das obras de arte, depois de um breve toque com o dispositivo móvel por parte dos utilizadores, desencadeia o arranque da aplicação, o que faz com que seja exibido o conteúdo armazenado na *tag*. Neste modo de funcionamento, se a *tag* NFC possuir capacidade para armazenar toda a informação desejada, não é necessária a utilização de outro protocolo de comunicação. Se a capacidade da *tag* não for suficiente, é colocado um resumo da informação, e é dada ao utilizador a hipótese da obtenção do restante conteúdo utilizando a ligação *Bluetooth* ou HTTP.

Utilizando um dispositivo NFC com interface USB, é possível entregar conteúdos de qualquer dimensão a partir de um computador, utilizando apenas o protocolo NFC-IP1, tal como na aplicação de modo quiosque do exemplo de implementação apresentado.

4.1 Modelo de distribuição da aplicação de visualização de conteúdos

As descrições anteriores partem do princípio que a aplicação já se encontra instalada no dispositivo do utilizador. Quando este não for o caso, existe um passo adicional que passa pela instalação da aplicação.

São suportadas várias formas de distribuição:

- Instalação manual via *Bluetooth*
  - O utilizador dirige-se a um quiosque ou a um serviço de atendimento que tenha sido criado para dar suporte ao serviço de distribuição de conteúdos.
- Instalação manual através da Internet
  - Procedendo ao download da *midlet*, ao navegar num sítio *Web*.

    Caso o utilizador esteja a utilizar um computador, pode efectuar o download da aplicação e instalar no seu dispositivo móvel, utilizando um cabo de dados ou uma ligação sem fios.

    Se o utilizador proceder à navegação a partir de um dispositivo móvel que suporte a tecnologia *Java ME*, pode proceder à instalação directamente a partir da Internet. Para isto ser possível, basta que do lado do servidor esteja montado um sistema de *Over-The-Air* (OTA) *Provisioning*. Este é um tipo de sistemas que permite o *download* de aplicações *Java ME* a partir de dispositivos com ligação à Internet. Opcionalmente, estes sistemas podem detectar qual o dispositivo que se encontra a efectuar o pedido da aplicação[1] e entregar a versão correcta da aplicação para o dispositivo em causa. Este mecanismo de instalação será descrito com maior detalhe na Secção 5.6.5.1.
- Instalação automática utilizando informação armazenada em *tag* NFC
  - Ao tocar na *tag* NFC, o utilizador é questionado se deseja efectuar o *download* e instalação da aplicação via GPRS ou *Wireless Ethernet*. Este cenário de instalação automática não é aplicável para o caso em que o meio de entrega de conteúdo é uma ligação NFC-IP1. Neste caso, a aplicação já necessita de se encontrar em execução antes da interacção com o outro equipamento NFC.

    Neste ponto importa referir que, caso seja tocada uma *tag* com conteúdo criado na aplicação de edição de conteúdos e a aplicação não esteja instalada, é possível desencadear a instalação da mesma.

    Para suportar este funcionamento, as *tags* têm de ser gravadas de forma adequada para que seja desencadeado o processo de instalação da *midlet*. Os detalhes relacionados com esta forma de distribuição da aplicação são apresentados na Secção 5.6.5.2.

[1] Tipicamente a identificação é feita com recurso ao cabeçalho HTTP *User Agent* [14] enviado pelo *browser* do dispositivo móvel, quando é feito o pedido HTTP GET.

4.2 Cenários de utilização

Na Figura 16 são apresentados alguns dos cenários de utilização que são implementáveis recorrendo aos componentes actualmente existentes.

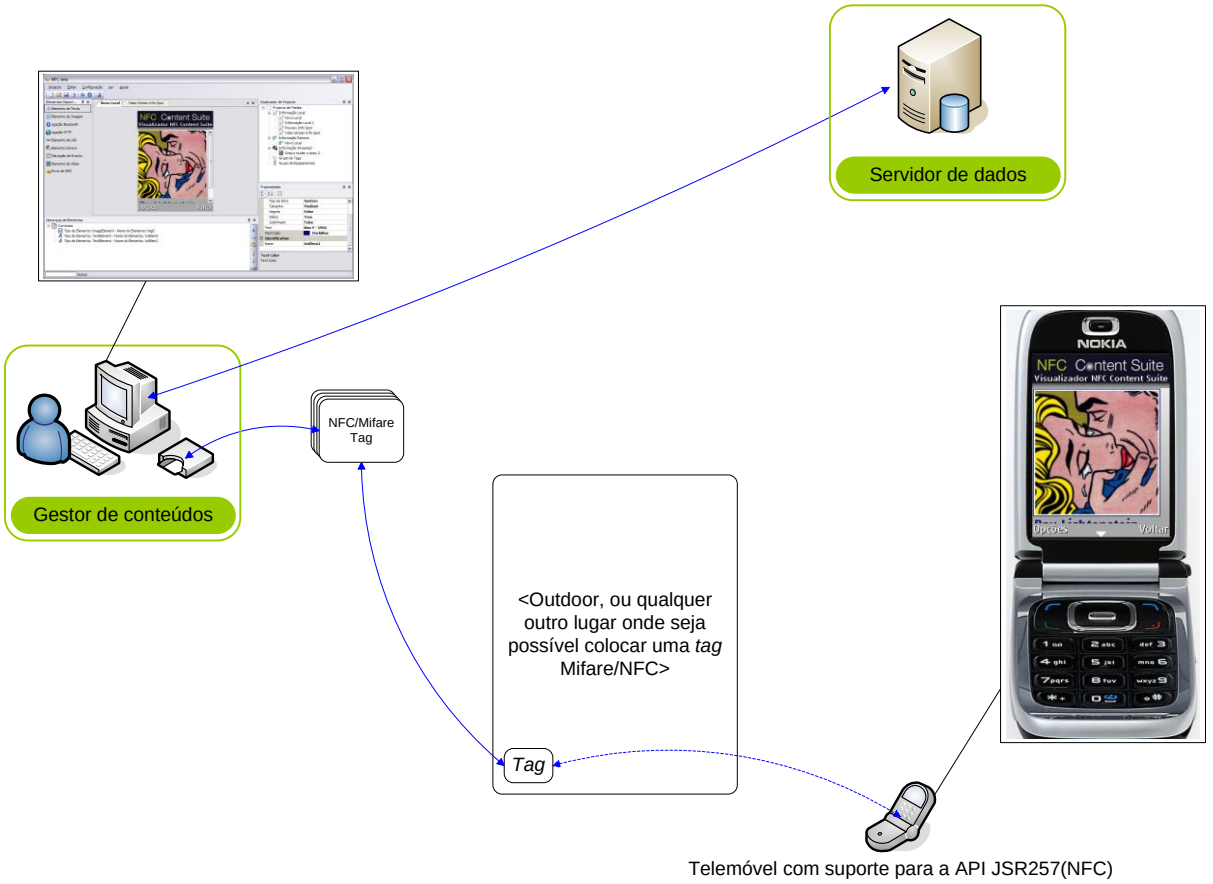


Figura 16 – Cenário de utilização – Todo o conteúdo cabe na *tag*

No cenário ilustrado na Figura 16, é apresentada a utilização da solução desenvolvida para o exemplo da distribuição de informação em *outdoors*. As *tags* NFC encontram-se coladas na publicidade em papel ou nas próprias estruturas onde é afixada a publicidade. No primeiro caso, são utilizadas etiquetas descartáveis, previamente gravadas. Para suportar a segunda hipótese, teriam de ser utilizadas *tags* resistentes a vandalismo. Os conteúdos presentes nestas *tags* seriam actualizados quando fosse feita a distribuição da publicidade em papel, através da utilização de uma aplicação instalada num terminal NFC. Este cenário de aplicação só é possível nos casos em que a capacidade da *tag* NFC utilizada é suficiente para o conteúdo em causa. Após a escrita, realizada através da aplicação de edição de conteúdos, as *tags* podem ser distribuídas pelos vários meios, e não é necessária qualquer ligação a infra-estruturas externas.

Este é o cenário mais simples, porque não envolve outras comunicações além da leitura da *tag* e, simultaneamente, o mais intuitivo, uma vez que basta encostar o dispositivo à *tag* NFC para que o conteúdo seja exibido.

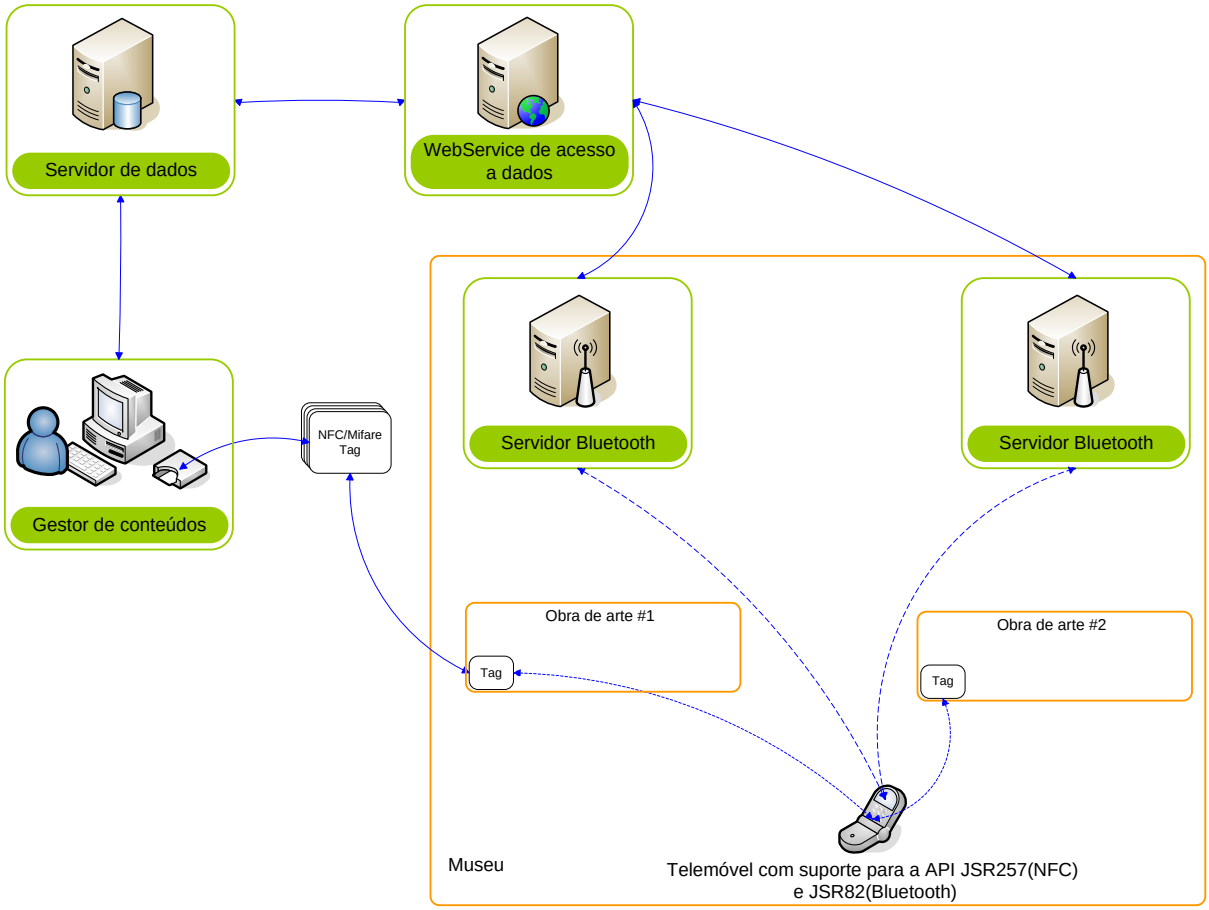


Figura 17 – Cenário de utilização – Parte do conteúdo na *tag*, restante obtido através de *Bluetooth*

Para conteúdo mais complexo, em que não é possível colocar toda a informação nas *tags*, e em cenários em que seja possível gerir uma infra-estrutura *Bluetooth*, é possível obter os dados adicionais através de uma ligação *Bluetooth* a servidores dispersos pelo local da instalação da solução, como ilustrado na Figura 17. Desta forma, o utilizador não incorre em qualquer custo de comunicações. Neste cenário, é apresentada uma implementação deste sistema num museu onde a conectividade adicional que é necessária para alguns conteúdos é suportada apenas por ligações *Bluetooth*. Os servidores de conteúdos *Bluetooth* encontram-se dispersos pelas instalações do museu para que os utilizadores possam obter informação em toda a área de exposição. Tipicamente, é necessária uma acção do utilizador para que seja efectuada a ligação ao servidor de conteúdos para obter informação adicional. No entanto, quando são utilizados elementos de ligação e não se deseja colocar informação adicional na *tag* NFC, é desencadeada a ligação ao servidor de conteúdos de forma automática. Isto é feito através da configuração de uma propriedade do elemento de ligação. Desta forma, logo após o carregamento do elemento de ligação a partir da *tag* NFC, é efectuada a ligação para obtenção de informação.

Uma vantagem em relação aos sistemas de distribuição de conteúdos via *Bluetooth* actuais, é que o utilizador recebe apenas a informação que lhe interessa, porque necessita de pedir essa mesma informação explicitamente através do toque do dispositivo móvel numa *tag*.

Outra vantagem é que do ponto de vista do protocolo *Bluetooth*, o dispositivo do utilizador não necessita de estar visível para os outros, nem de aceitar objectos remotos. Este facto configura uma vantagem porque caso contrário, utilizadores mal-intencionados (ou simplesmente alguém que se encontre nas redondezas

do local do ponto de acesso *Bluetooth*), podem "ver" os dispositivos dos utilizadores e enviar objectos prejudiciais (vírus/*worms*) por este meio.

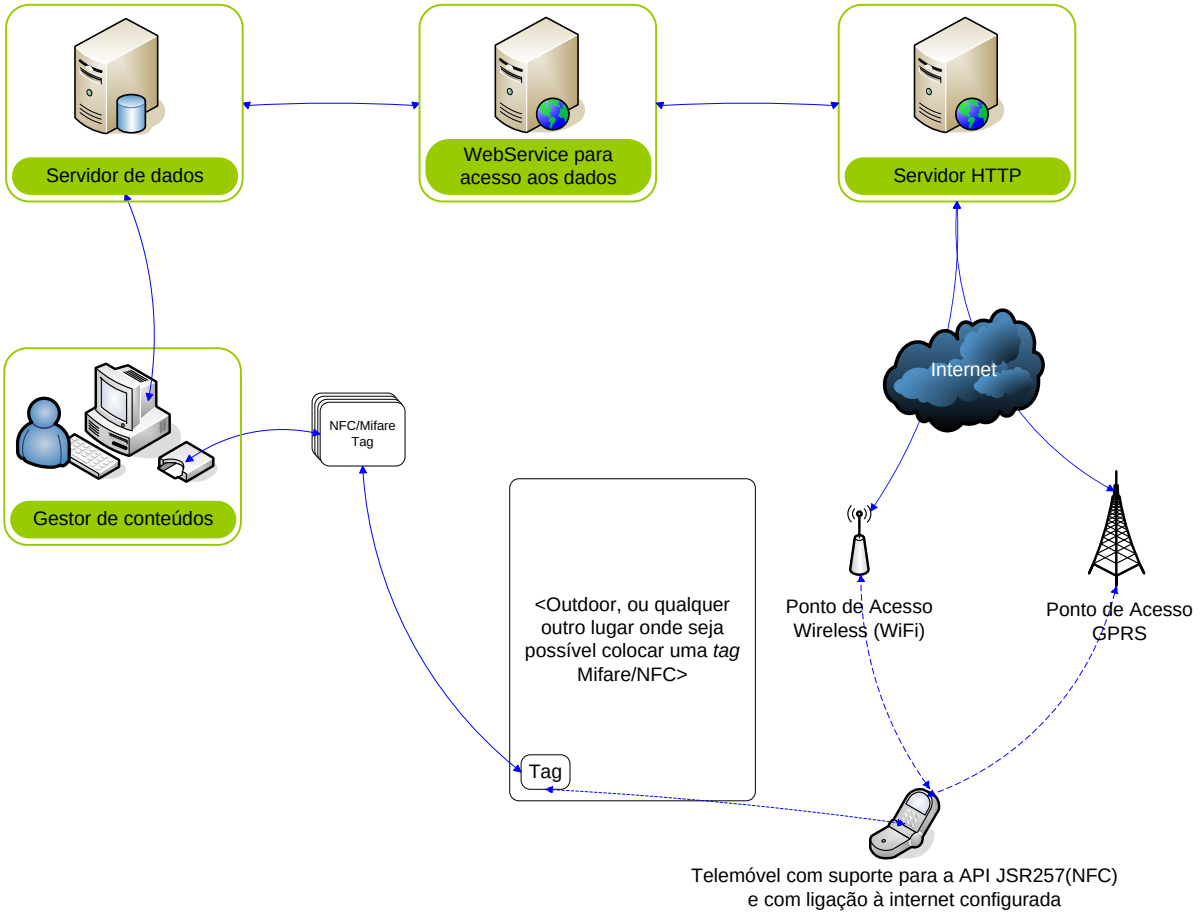


Figura 18 – Cenário de utilização – Parte do conteúdo na *tag*, restante obtido através de HTTP

A existência de um servidor HTTP levanta a possibilidade da entrega de conteúdo utilizando qualquer ligação à Internet que o utilizador tenha configurado no dispositivo. Assim, é possível suportar a entrega de conteúdo utilizando infra-estruturas de rede *Wireless Ethernet* que possam já existir, tal como no exemplo ilustrado na Figura 18. Neste exemplo, é elaborado o cenário de publicidade em *outdoors* apresentado na Figura 16, adicionando a possibilidade de o conteúdo ser obtido utilizando uma ligação à Internet. Neste cenário, a *tag* contém apenas um documento com um único elemento de ligação que é utilizado para obter o conteúdo da campanha publicitária associada. As *tags* são gravadas antes de serem colocadas nos devidos locais, e não necessitam de ser alteradas, uma vez que a gestão do conteúdo a ser apresentado pode ser efectuada na aplicação de edição de conteúdos. Neste caso, o elemento de ligação é configurado para se ligar automaticamente após o carregamento do documento a partir da *tag* NFC.

Nos casos em que não é possível gerir uma infra-estrutura *Bluetooth* ou *Wireless*, (quer por dispersão geográfica, por motivos administrativos, ou outros), a solução passa pela utilização da rede celular para a obtenção do restante conteúdo. Aqui deve ser tomada especial atenção à dimensão do conteúdo, uma vez que podem ser cobradas ao utilizador as comunicações efectuadas (dependendo do seu tarifário do plano de dados). Esta solução tem o inconveniente adicional de exigir que o utilizador tenha o seu dispositivo móvel previamente configurado para acesso GPRS.

Esta solução tornar-se-á mais interessante no momento em que este tipo de acesso passe a ser muito comum, cenário esse em que os custos envolvidos também terão tendência para reduzir.

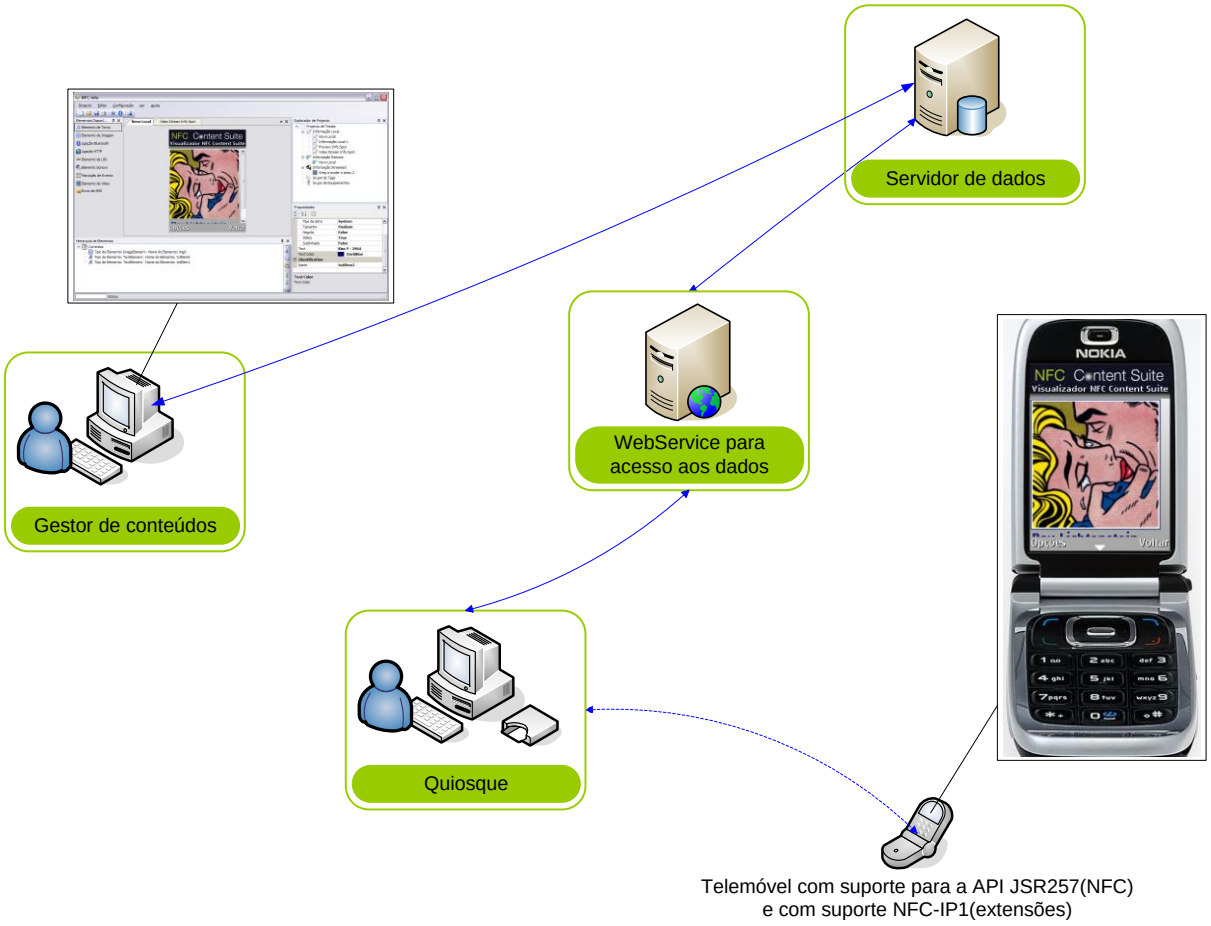


Figura 19 – Cenário de utilização – Transmissão de conteúdo através do protocolo NFC-IP (*Peer to Peer*)

Utilizando um dispositivo NFC com interface USB e suporte para o protocolo NFC-IP1, é possível obter informação a partir de computadores pessoais. Tirando partido deste facto, para além dos conteúdos previamente descritos que são definidos através da ferramenta de criação de conteúdos, pode ser criada uma solução em que o conteúdo a enviar para o dispositivo móvel é gerado de forma dinâmica e/ou personalizada, tal como ilustrado na Figura 19.

Uma aplicação possível destas capacidades é a criação de um quiosque electrónico, em que os utilizadores interagem com uma aplicação de configuração de extras de viaturas de determinado fabricante. Nessa aplicação é possível escolher os extras e as características dos veículos. Finalmente, o utilizador pode guardar as configurações escolhidas no seu dispositivo móvel com capacidades NFC, sendo o envio do conteúdo efectuado através da utilização de um leitor/escritor NFC com suporte para o protocolo NFC-IP1. O utilizador pode mais tarde consultar calmamente as escolhas que efectuou, bem como os preços finais de cada configuração. O conteúdo armazenado pode ainda ser utilizado para a indicação a um representante da marca de qual o modelo e opções pretendidas.

Figura 20 – Cenário de utilização – Transmissão de conteúdo através do protocolo NFC-IP entre dois telemóveis

Outro cenário possível é a troca de conteúdo entre utilizadores. Neste cenário, ilustrado na Figura 20, o utilizador selecciona o conteúdo que deseja enviar (obtido previamente através de qualquer uma das formas de obtenção de conteúdo disponíveis), aproxima os telemóveis e a transferência é realizada. Esta funcionalidade pode ser utilizada pelos utilizadores do cenário ilustrado na Figura 19, para partilhar as configurações efectuadas na aplicação de configuração de viaturas.

4.3 Sumário do capítulo

Neste capítulo foram apresentados os componentes que compõem a aplicação e a forma como estes interagem entre si. Para isso, foram utilizados alguns dos cenários que podem ser criados através da utilização desses componentes.

A arquitectura actual permite a criação de soluções que podem ir desde a entrega de conteúdos através da consulta de uma *tag*, passando por cenários em que são utilizados servidores *Bluetooth* e HTTP, até à utilização do protocolo NFC-IP1 como meio de obtenção de informação. A arquitectura definida permite a utilização em conjunto de todas estas formas de comunicação.

Finalmente, foram também identificadas as formas como a aplicação de visualização de conteúdos pode ser instalada nos terminais dos utilizadores.

Capítulo 5

5 Implementação

No presente capítulo são apresentados os principais aspectos de implementação dos componentes da solução desenvolvida. O capítulo começa com a descrição dos componentes relacionados com o servidor de dados, sendo de seguida apresentada a forma como as aplicações interagem com esse servidor.

Antes da apresentação da aplicação de edição de conteúdos, que permite a edição e gestão dos conteúdos que são enviados para os utilizadores da solução, é introduzido o motor de seriação de dados. Este é utilizado pela aplicação referida anteriormente, e implementa o modelo de transferência de dados apresentado na Secção 4.1.

O capítulo prossegue com a apresentação das bibliotecas desenvolvidas para a comunicação com leitores NFC. Estas permitem a escrita de *tags* Mifare com mensagens NDEF e o envio de informação para dispositivos que suportem o protocolo NFC-IP1.

Seguidamente, é efectuada a apresentação dos principais aspectos da aplicação de visualização de conteúdos em dispositivos móveis.

Finalmente, é efectuada a descrição das implementações dos servidores de conteúdos desenvolvidas para esta solução. Para suportar esta descrição, é previamente apresentado o *Web service* para acesso ao servidor de dados, bem como quais os motivos que levaram à sua criação.

5.1 Servidor de dados

A camada de persistência é suportada por uma base de dados *SQL Server 2005*. Esta camada dá suporte de armazenamento persistente não só para os documentos a serem enviados para os utilizadores da solução, mas também para informações relativas ao contexto em que esses documentos serão exibidos. Nesta secção são apresentados os aspectos mais relevantes da camada de persistência, cuja organização pode ser consultada na íntegra no Anexo 7.2.

Ao local onde é colocada uma *tag* NFC gravada para funcionar com este sistema, foi dado o nome de ponto de informação. Cada ponto de informação possui identificador único e tem associado o conteúdo que é exibido nessa localização. A localização pode ser caracterizada através de uma descrição, identificando por exemplo obras de arte num museu.

Existem três tipos de pontos de informação: locais, remotos e multimédia.

Os pontos de informação local são aqueles cujo conteúdo é obtido unicamente a partir de uma *tag* NFC. Estes podem conter toda a informação pretendida, ou possuir elementos de ligação que obtêm informação efectuando pedidos a um servidor de conteúdos. Os conteúdos obtidos desta forma estão associados a pontos de informação remota. Os pontos de informação multimédia são utilizados para guardar o conteúdo de ficheiros de áudio e vídeo cujo conteúdo é reproduzido no dispositivo, utilizando o suporte multimédia fornecido pela *Mobile Media* API (MMAPI)[15]. As razões que levam a esta divisão são identificadas na Secção 5.6.8.1.

Os pontos de informação são organizados em projectos, que contêm também outras informações tais como descrição do projecto, validade, cliente e localização. Apesar de actualmente apenas serem suportados os dispositivos Nokia 6131NFC[16] e 6212 Classic[17], foi incluído o suporte para o registo da informação acerca dos terminais-alvo de cada projecto. Isto pode ser utilizado mais tarde para alterar o modo de edição na aplicação de edição de conteúdos. Cada projecto tem também associado um conjunto de *tags* NFC, com as respectivas características técnicas. Estas informações são importantes para vir a suportar a identificação de quais as *tags* que possuem capacidade suficiente para gravar a informação associada a determinado ponto de informação local.

Para cada ponto de informação é mantido um histórico das acções que são realizadas sobre o seu conteúdo. Cada entrada no histórico de um ponto de informação possui a informação associada, guardada na forma de uma sequência de bytes. Esta organização é ilustrada na Figura 21.

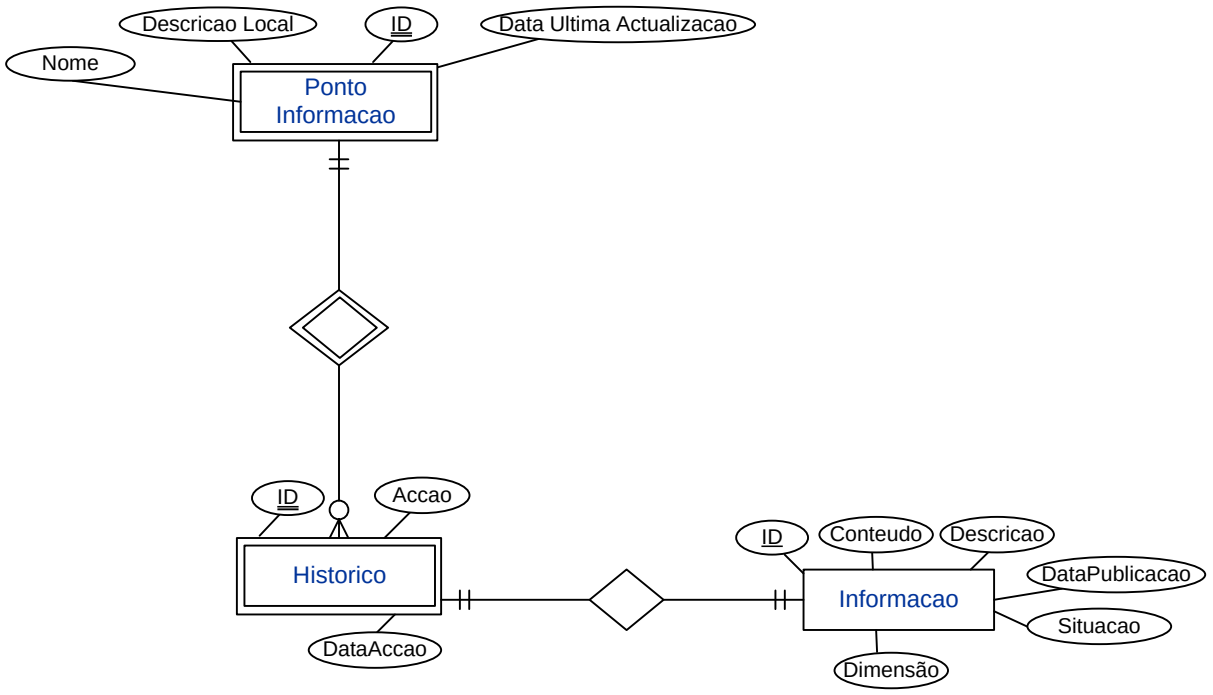


Figura 21 – Excerto do modelo E-A relacionado com o armazenamento dos conteúdos

5.2 Camada de acesso a dados

A camada de acesso a dados fornece acesso de forma simplificada aos dados armazenados na camada de persistência. Esta camada permite obter instâncias de classes que representam a informação de entidades armazenadas na base de dados.

Tipicamente uma tabela presente na base de dados é mapeada numa classe. Da mesma forma, as colunas da tabela são mapeadas em campos ou propriedades nessa mesma classe. Este mapeamento não é válido para as tabelas que são criadas durante o processo de normalização para suportar relações entre tabelas. Este tipo de relações, dependendo da multiplicidade da relação em causa pode ser suportado por colecções de objectos ou por uma propriedade que refere outro objecto. Apesar de ser um mapeamento que serve o propósito da transferência de informação, perde-se a capacidade de efectuar interrogações à base de dados. Isto pode ser resolvido através da criação de métodos de pesquisa nas classes que resultam do mapeamento. Apesar de em alguns casos, este

passo ser relativamente simples, como é o caso da obtenção de todas as instâncias, ou de uma instância particular a partir de um identificador, surgem por vezes interrogações que têm de ser criadas de forma manual.

Existem várias ferramentas, utilizando diferentes abordagens, que abordam este problema. A este conjunto de ferramentas é dado o nome de ferramentas de mapeamento objectos-relacional (*Object-Relational Mapping*), ou ORM.

Na maior parte dos casos, após o processamento do modelo relacional por parte da ferramenta de ORM, é gerado o conjunto de classes a ser utilizado para aceder aos dados armazenados na base de dados, sem necessidade de conhecer os detalhes desse acesso. O mapeamento pode ser feito de várias formas, tais como ficheiros de configuração ou de forma automática pela ferramenta ORM. No caso deste projecto, é utilizada a ferramenta de ORM *Language INtegrated Query* (LINQ) to *Structured Query Language* (SQL)[18], da Microsoft.

O mapeamento entre o modelo da base de dados e o modelo de objectos é feito com recurso a uma ferramenta, integrada no ambiente de desenvolvimento (*Visual Studio 2008*), que obtém o esquema da base de dados seleccionada. Depois deste passo, a ferramenta gera o conjunto de classes. Aqui, são inferidas as multiplicidades das relações a partir do esquema. Quando existem relações de N para M, é criada uma classe que relaciona as entidades em questão, à semelhança do que acontece no modelo relacional. A Figura 22 apresenta o mapeamento efectuado neste caso, entre as entidades no modelo E-A e os objectos *LINQ to SQL*.

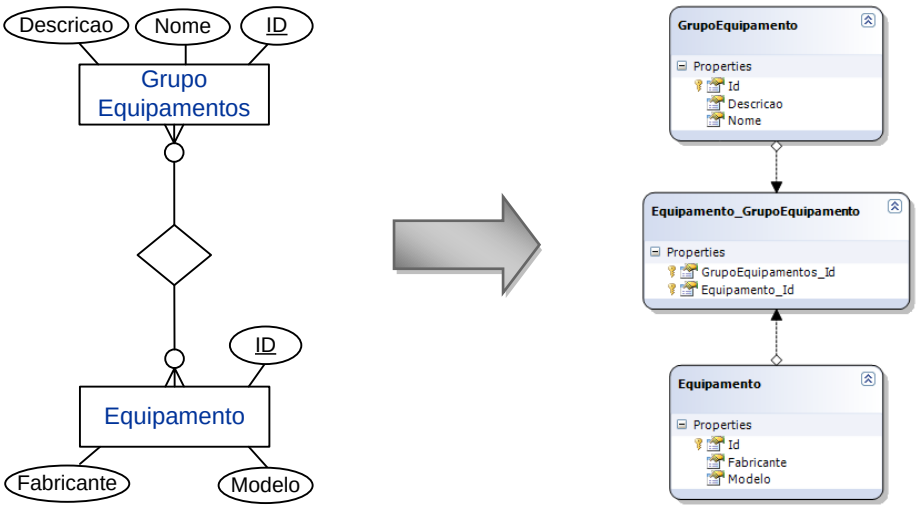


Figura 22 – Comparação entre o modelo Entidade-Associação e o modelo de objectos criado pela ferramenta de ORM (caso N:M)

A utilização destes objectos começa pela instanciação do *DataContext*. O *DataContext* traduz os pedidos de objectos sobre ele realizados, efectua as interrogações SQL necessárias e cria instâncias dos objectos pedidos a partir dos resultados das interrogações. A ferramenta de geração automática das classes cria também uma classe que estende o tipo *DataContext*, adicionando membros específicos para a base de dados em causa. Na instanciação do *DataContext* é especificada a *string* de ligação à base de dados. Este objecto contém propriedades que representam as tabelas existentes na base de dados. Estas propriedades têm o nome das tabelas associadas na base de dados, mas no plural. O Exemplo 1 ilustra a utilização destas propriedades para a obtenção de todos os projectos cujo nome possua uma determinada sequência de caracteres.

Exemplo 1 – Obtenção de todos os projectos filtrados pelo seu nome, através do *DataContext*

```
IEnumerable<Projecto> projectos = BackEnd_DataContext.Projectos;
return from p in projectos
       where p.Nome == null || p.Nome.Contains(name)
       select p;
```

A navegação no modelo gerado é efectuada através de propriedades. No Exemplo 2 é ilustrada a navegação no modelo criado, através da utilização de entidades de relação que funcionam como as tabelas de relação no modelo relacional. Estas tabelas são geradas para o caso em que a relação é do tipo N:M.

Exemplo 2 – Utilização das classes geradas pela ferramenta de ORM (caso N:M)

```
foreach (Projecto_GrupoEquipamento pge in projecto.Projecto_GrupoEquipamentos)
{
       String nomeGrupoEquipamentos = pge.GrupoEquipamento.Nome;
       foreach (Equipamento_GrupoEquipamento ege in
              pge.GrupoEquipamento.Equipamento_GrupoEquipamentos)
       {
              String fabricante = ege.Equipamento.Fabricante;
              String modelo = ege.Equipamento.Modelo;
       }
}
```

Caso a associação seja de 1 para 1 ou 1 para N, é criada uma propriedade com o nome da tabela relacionada. Quando existem associações de 1 para N é também criada uma propriedade com o nome da tabela relacionada, mas no plural.

A Figura 23 apresenta o mapeamento efectuado pela ferramenta de ORM.

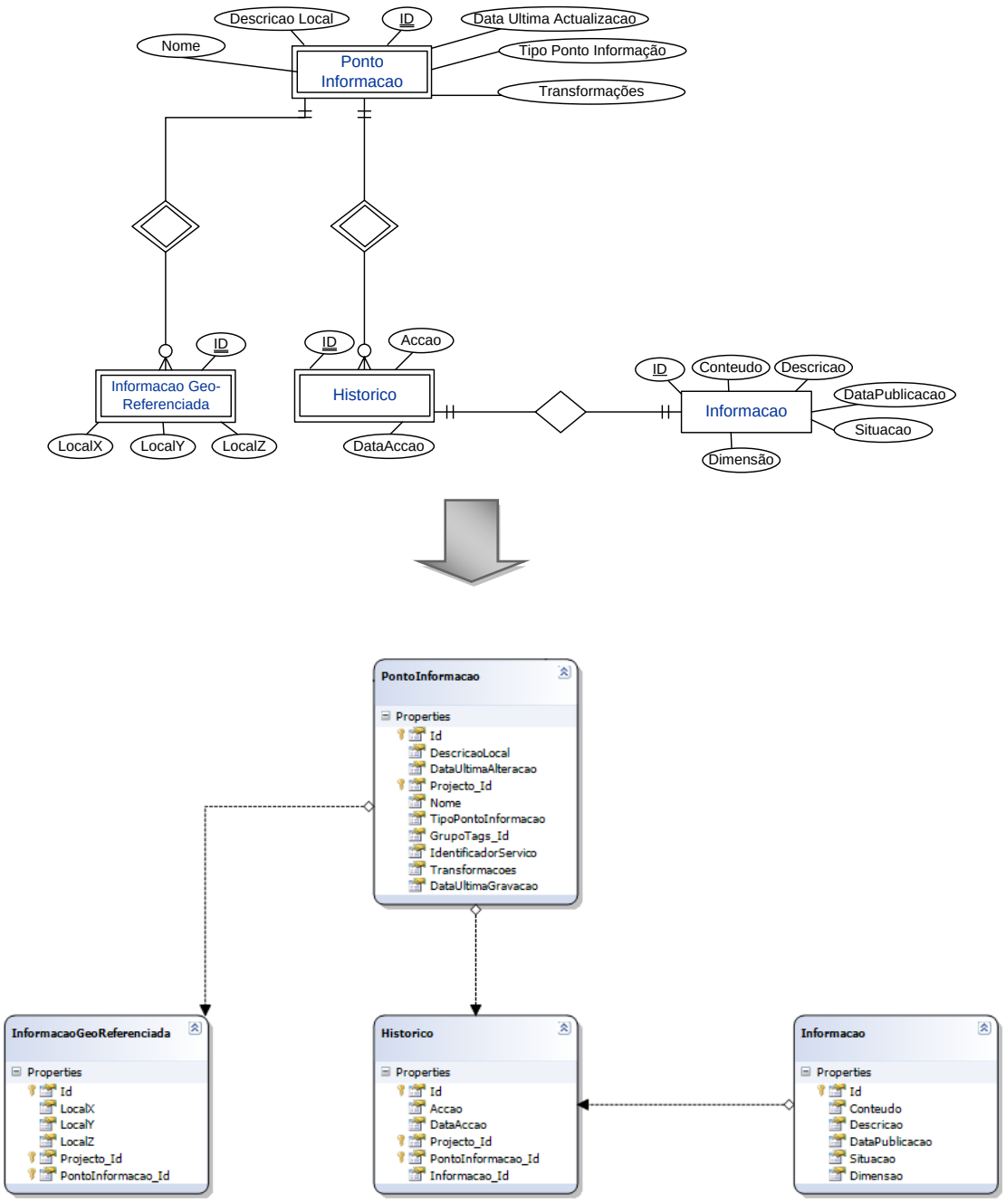


Figura 23 – Comparação entre o modelo Entidade-Associação e o modelo de objectos criado pela ferramenta de ORM (casos 1:N e 1:1)

Nestes casos, a navegação no modelo é mais simples, uma vez que não é necessário navegar através dos objectos de relação.

O Exemplo 3 ilustra a utilização das classes geradas para estes casos.

Exemplo 3 – Utilização das classes geradas pela ferramenta de ORM (casos 1:N e 1:1)

```
byte[] data = informationSpot.Historicos.Last<Historico>().Informacao.Conteudo;
```

A diferença entre esta tecnologia e a maioria das ferramentas de ORM é que permite efectuar interrogações ao conjunto de objectos que é gerado para representar as tabelas que existem na Base de Dados. Estas interrogações são semelhantes às que são efectuadas usando a linguagem T-SQL. No entanto, como o

compilador de *C# 3.0* conhece os tipos dos objectos envolvidos na interrogação, as expressões envolvidas são também elas compiladas e verificadas quanto à sua validade. O Exemplo 4 apresenta a utilização desta funcionalidade.

Exemplo 4 – Interrogações através da utilização das funcionalidades da linguagem C# 3.0

```
Projecto projecto = BackEnd_DataContext.CurrentProject;
IEnumerable<PontoInformacao> pis =
    from pontoInformacao in projecto.PontoInformacaos
    where ((name == String.Empty ||
           (pontoInformacao.Nome.LastIndexOf(name) > 0)) &&
           (pontoInformacao.TipoPontoInformacao == (int)InformationSpotType))
    select pontoInformacao;
```

5.3 Motor de Seriação de dados

Para facilitar o envio e armazenamento de informação, independentemente do meio de transmissão ou persistência, foi desenvolvido um motor de seriação de dados. Este encarrega-se de transformar elementos de conteúdo em *bytes* e vice-versa, conforme exemplificado na Figura 24 e na Figura 25. Este motor utiliza o formato de seriação apresentado anteriormente.

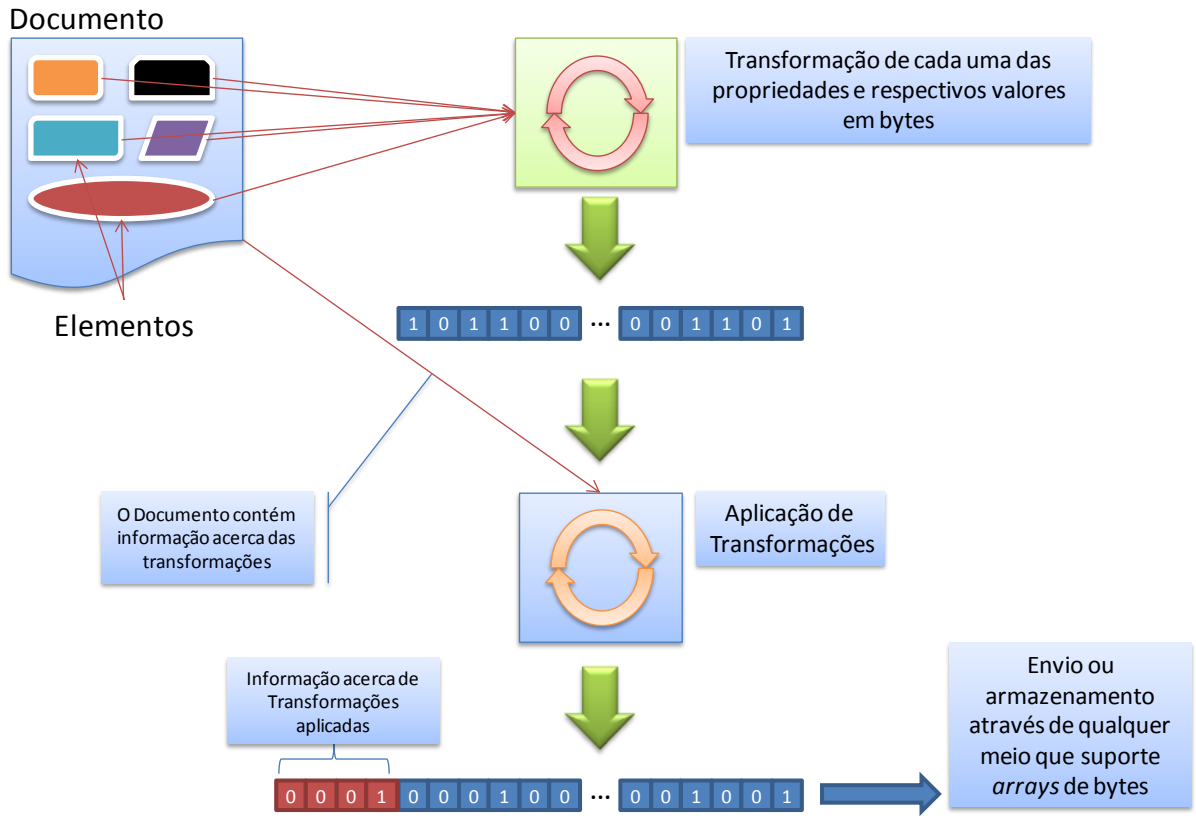


Figura 24 – Diagrama do funcionamento da seriação dos conteúdos

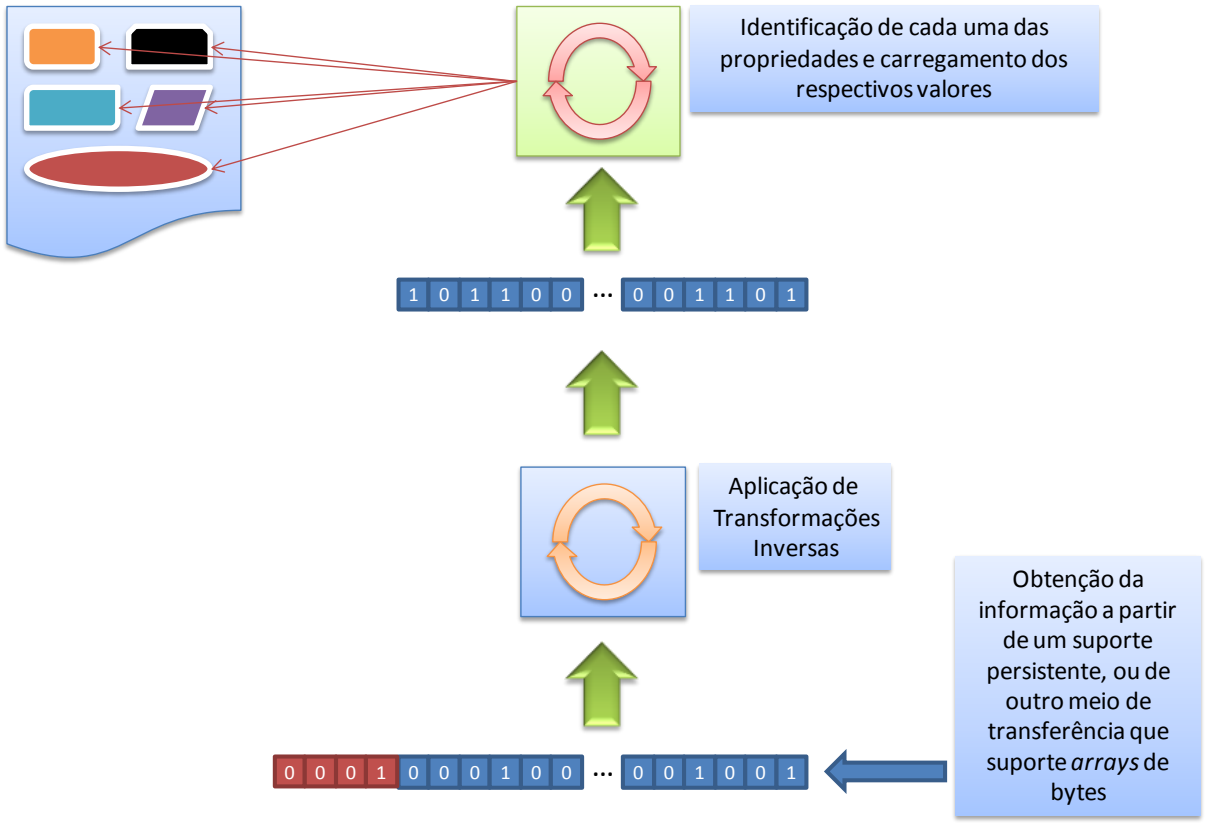


Figura 25 – Diagrama do funcionamento da deseriação dos conteúdos

O motor de seriação recorre à funcionalidade de informação de tipo disponível em tempo de execução da plataforma .NET, através da utilização dos tipos existentes no espaço de nomes `System.Reflection`.[19] Utilizando esta funcionalidade, é possível automatizar grande parte do processo de seriação do conteúdo.

Os tipos que representam os elementos disponíveis são marcados com atributos que possuem a informação necessária ao processo de seriação.

Exemplo 5 – Excerto de um elemento e de uma propriedade desse elemento, anotados com atributos

```
[ToolCodeMapping(ToolCode.ImageElement)]
[DefaultName("img{0}")]
public class ImageElement : MediaElement
{
   private String text;
   [SendToTarget]
   [DefaultPropertyValue(typeof(String), "Empty")]
   [PropertyNameMapping(PropertyConstants.Text)]
   [PropertyFormat(FieldFormat.UnicodeStringBE)]
   [LocalizedCategory("CategoryContent")]
   [LocalizedDescription("ImageElement_Text_Description")]
   [LocalizedDisplayName("ImageElement_Text_DisplayName")]
   [Editor(typeof(MultilineStringEditor), typeof(UITypeEditor))]
   public String Text
   {
      get
      {
          return text;
      }
      set
      {
          text = value;
          OnPropertyChanged();
      }
   }
}
```

De seguida são apresentadas algumas das características do motor de seriação que são suportadas com recurso a estas funcionalidades.

Para reduzir a quantidade de informação gerada através do processo de seriação, todas as propriedades possuem um valor por omissão. Se quando for efectuada a seriação do elemento, o valor de determinada propriedade for igual ao valor por omissão, esta propriedade não é seriada.

O valor por omissão de cada propriedade é definido através da aplicação de um atributo do tipo `DefaultPropertyValueAttribute`.

A forma como cada propriedade é seriada varia consoante o tipo de dados que armazena. Para este efeito, podia ser utilizada a informação de tipo associada à propriedade. No entanto, em certos casos essa informação não é suficiente, tal como no caso das *strings*, dado que uma propriedade deste tipo pode ser seriada de formas diferentes, dependendo da semântica associada à propriedade. Por exemplo, podemos tomar o caso de uma *string* de ligação *Bluetooth*, para a qual pode ser utilizada a codificação de caracteres UTF-8. No entanto, nos casos em que seja exibido texto ao utilizador, é desejável que seja suportada a codificação Unicode.

Poderia ter sido tomada a decisão de apenas suportar Unicode, mas essa opção provocaria o aumento desnecessário da dimensão do resultado da seriação. Quando comparada com UTF-8, a codificação em Unicode pode exigir o dobro dos bytes para representar a mesma *string*[1].

Para evitar este problema, todas as propriedades são marcadas com um atributo do tipo `PropertyFormatAttribute`, que identifica a forma como é feita a seriação do conteúdo da propriedade.

Uma vez que nem todas as propriedades são seriadas, todas elas têm de ser identificadas aquando da sua seriação. Essa identificação é feita com recurso a um byte que funciona como um identificador. De forma a manter o processo de seriação o mais automatizado possível, todas as propriedades possuem um atributo do tipo `PropertyNameMappingAttribute` onde é definido qual o identificador da propriedade em questão.

Outra das funcionalidades que é suportada através de atributos é a exclusão de uma propriedade do processo de seriação. Isto é útil para casos em que o valor da propriedade pode ser obtido à custa de outras propriedades ou através de outra informação disponível quando é feito o processo inverso. Isto é conseguido mais uma vez através da aplicação de um atributo, do tipo `SendToTargetAttribute`, que indica ao motor de seriação que a propriedade em causa não deve ser seriada.

5.3.1 Suporte adicional para o editor gráfico

Para além das funcionalidades anteriormente descritas, os elementos são anotados com outros atributos, que auxiliam a representação gráfica dos mesmos, no editor gráfico que será apresentado na Secção 5.4.

Quando um elemento é adicionado a um documento deve possuir um nome. O formato desse mesmo nome é obtido a partir do valor do atributo `DefaultNameAttribute`. Este atributo é aplicado ao tipo do elemento, e o nome final é dado pela substituição do marcador de posição (identificado pela marcas "{0}"), pelo índice dentro do documento actual. No caso do elemento apresentado no Exemplo 5, quando são adicionados 2

[1] Dependendo dos caracteres a codificar, esta codificação pode até ocupar mais do que a codificação Unicode. No entanto, as propriedades que são representadas com esta codificação apresentam apenas caracteres que tipicamente se encontram entre os valores Unicode U+0000 e U+007F (compatível com ASCII), o que resulta neste caso, em metade da dimensão ocupada em relação à codificação Unicode.

elementos deste tipo e não é alterado o nome de nenhum deles, estes ficarão com os seguintes nomes: img1 e img2.

Em determinados estados da aplicação, tal como a criação de controlos gráficos para associação a elementos, é necessário saber qual o tipo de controlo que está associado a determinado tipo de elemento. Esta funcionalidade é suportada através da marcação do tipo do elemento com um atributo do tipo `ToolCodeMappingAttribute`.

A edição das propriedades dos conteúdos é suportada através da utilização de uma instância do controlo `PropertyGrid`. Esta permite a edição de vários tipos de dados sem intervenção do programador. Através da navegação das propriedades do objecto que se encontra a ser editado, este controlo consegue fornecer capacidades de edição para um número elevado de tipos de dados. Esta navegação é possível devido à existência de informação de tipo disponível em tempo de execução.[19] No entanto, em certos tipos de dados, é necessário indicar uma forma alternativa de definição do valor da propriedade. Isto é conseguido através da associação de um editor de tipo, através da utilização do atributo `EditorAttribute`, para o qual o controlo `PropertyGrid` delega a responsabilidade da alteração do valor dessa propriedade. Adicionalmente, a `PropertyGrid` suporta as seguintes funcionalidades:

- Divisão das propriedades em categorias.
    - Tipicamente esta divisão é suportada através da aplicação de atributos do tipo `CategoryAttribute` às propriedades desejadas. Este atributo não permite definir valores que variem consoante o idioma seleccionado. Para o tornar possível, foi criada uma especialização deste atributo para suportar esta funcionalidade (`LocalizedCategoryAttribute`). A cadeia de caracteres recebida no atributo contém a chave para o ficheiro de *resources* onde se encontra o mapeamento entre esta chave e o valor a exibir. O ficheiro de onde é extraído este mapeamento é carregado de acordo com a cultura actual.
- Alteração do nome de exibição de propriedades.
    - Por omissão o nome associado a uma propriedade é o nome do membro associado. Em certos casos, porém, pode ser interessante apresentar um nome mais apelativo, com espaços e outros caracteres que não podem pertencer ao nome da propriedade usado na *Common Language Infrastructure* (CLI). Pelos motivos identificados anteriormente, também foi necessário, para o caso do atributo `PropertyNameAttribute`, criar uma versão que suporte a localização automática do valor do atributo (`LocalizedPropertyNameAttribute`).
- Atribuição de descrição à propriedade.
    - De forma a ajudar os utilizadores a compreender qual o significado de determinada propriedade de um objecto, é possível atribuir uma descrição a cada propriedade. Tal como nos casos anteriores, onde normalmente seria utilizado um atributo do tipo `DescriptionAttribute`, é utilizado um atributo do tipo `LocalizedDescriptionAttribute`.

O resultado da utilização destes atributos será apresentado na Secção 5.4.

5.4 Editor gráfico para criação / gestão dos conteúdos

O editor foi criado utilizando a tecnologia *Windows Forms*. A Figura 26 apresenta o aspecto da janela principal da aplicação.

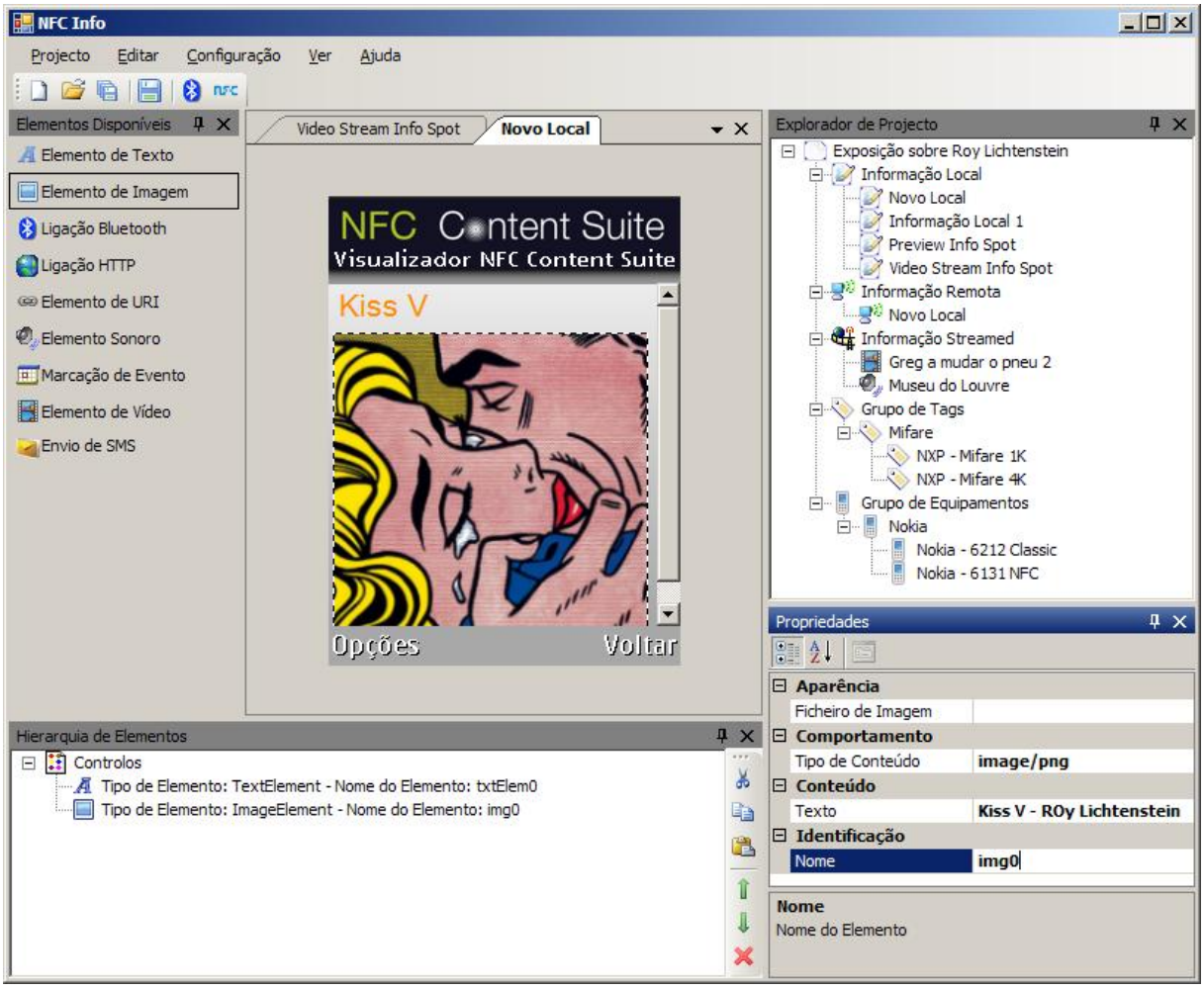


Figura 26 – Aspecto da aplicação de edição de conteúdos

A aplicação permite as seguintes operações:

- Criação / gestão de projectos.
- Criação / gestão de conteúdos (associado a um projecto).
    - Edição de todas as propriedades dos elementos, incluindo pré-visualização para conteúdos multimédia.
- Definição das transformações associadas a cada conteúdo.
- Interface optimizada para a gravação de número elevado de *tags*.
    - Esta funcionalidade permite a gravação de determinado conteúdo em várias *tags*.
- Cópia para ficheiro do conteúdo gerado.
- Envio do conteúdo actualmente em edição, por *Bluetooth* ou NFC-IP1.

5.4.1 Estrutura geral

A Figura 27 apresenta a estrutura geral da aplicação, bem como as interacções entre os seus principais componentes.

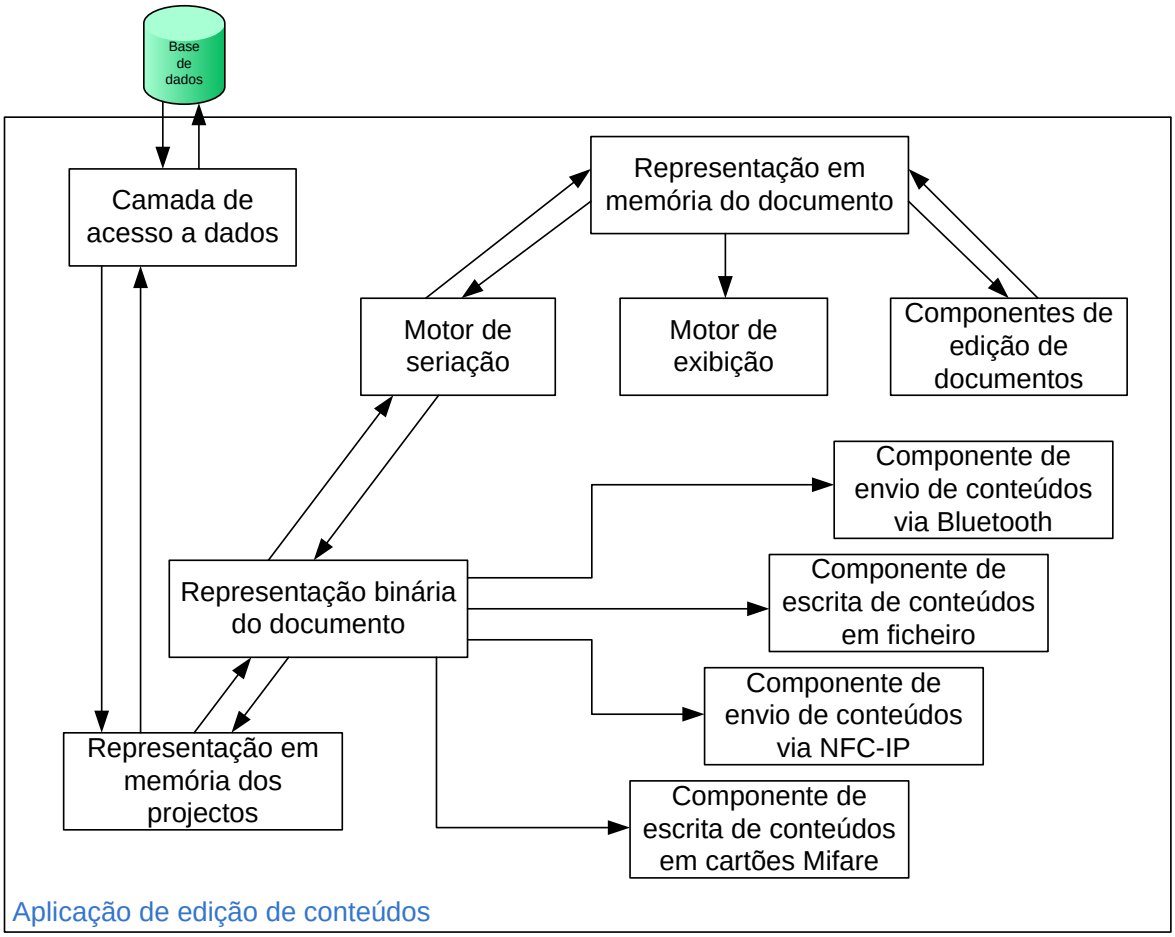


Figura 27 – Estrutura geral da aplicação de edição de conteúdos

De seguida são apresentados os componentes principais da aplicação, e é dada uma breve descrição dos papéis por eles desempenhados.

A camada de acesso a dados, apresentada na Secção 5.2, é responsável pela obtenção da informação a partir da base de dados e pela criação de objectos em memória que representem esses mesmos dados, no paradigma *object-oriented*. Esta representação em memória possui o conteúdo associado a determinado ponto de informação na forma de sequência de bytes. Para a obter a representação em memória do documento, é utilizado o motor de seriação que implementa o modelo de transferência de dados, apresentado na Secção 5.3. A representação em memória do documento é composta pelos vários elementos que compõem o documento associado a um determinado ponto de informação.

A representação em memória do documento obtida através da utilização do motor de seriação é depois utilizada pelo motor de exibição, para apresentar o conteúdo ao utilizador. O motor de exibição é baseado em controlos *Windows Forms*, e existe um tipo específico de controlo para cada tipo de elemento de informação. Os elementos de informação podem representar uma imagem, texto formatado, o envio de mensagem de SMS, entre outras. Estes controlos são depois colocados num contentor, que é responsável por dispor os controlos de forma adequada, da forma mais semelhante possível à que acontece na aplicação de visualização de conteúdos.

A edição dos documentos é suportada através de três tipos de componentes de edição de documentos, que se encontram agrupados de acordo com as seguintes funcionalidades: inserção de novos elementos de informação, edição de propriedades dos elementos e organização dos elementos dentro do documento.

Para além de ser mantido um histórico dos conteúdos editados, estes podem ainda ser armazenados em *tags Mifare* ou em ficheiros.

Para facilitar a o processo de edição de conteúdos, foi criada a possibilidade de enviar directamente o conteúdo que se encontra a ser editado para a aplicação de visualização de conteúdos. Esta funcionalidade é suportada através de ligações NFC-IP e *Bluetooth*.

De seguida serão apresentados alguns dos componentes que foram identificados na Figura 27.

5.4.2 Motor de exibição

O motor de exibição é composto por um controlo que age como contentor de elementos (`MobileCanvas`), e por controlos que estendem o tipo `NFCInfoElementControl`. Cada elemento tem associado um destes controlos, que são responsáveis pela exibição da informação contida no elemento.

Devido ao facto de existirem elementos cuja representação gráfica não varia de acordo com o valor das propriedades (como é o caso dos elementos de ligação e de áudio), foi criada a classe abstracta `ControlWithImage` que serve de base para todos os controlos que representam elementos com estas características. A hierarquia (parcial) de controlos criada para dar suporte à exibição de elementos é apresentada na Figura 28.

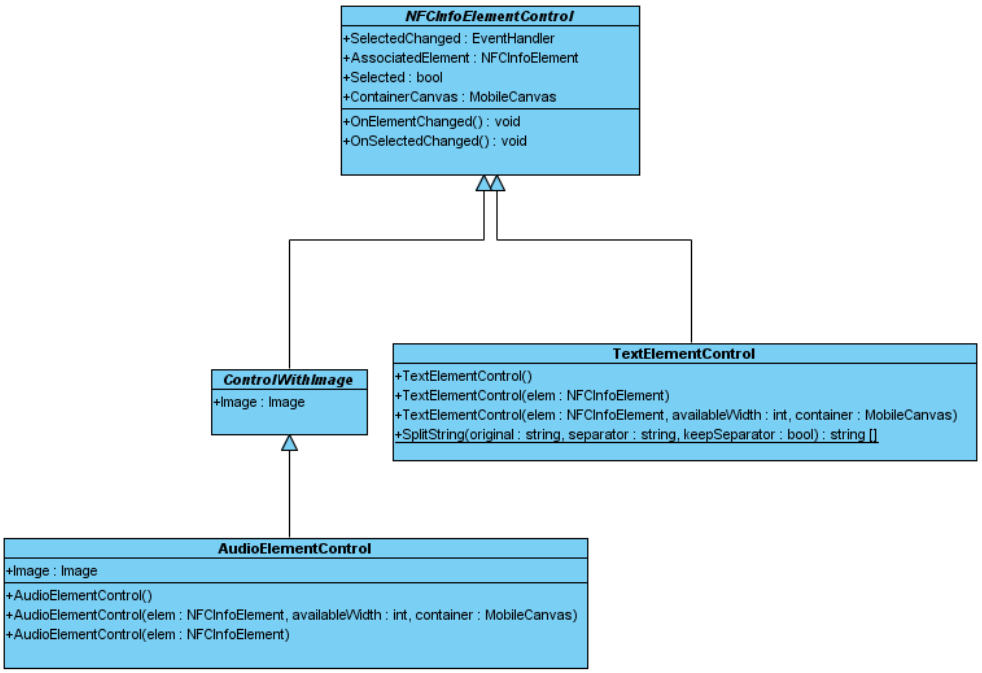


Figura 28 – Hierarquia (parcial) de controlos de exibição de elementos

A associação entre elementos e controlos é efectuada através da aplicação do atributo `ControlForElementTypeAttribute` às classes que implementam os controlos, como ilustrado no Exemplo 6.

Exemplo 6 – Utilização de atributos para associação de controlos gráficos a elementos de informação

```
[ControlForElementType(typeof(AudioElement))]
public partial class AudioElementControl : ControlWithImage{...}
```

A obtenção do controlo dado o elemento é feita através da utilização da classe `ControlFactory` apresentada na Figura 29. Esta classe implementa o padrão de desenho *FactoryMethod*[20].[21]

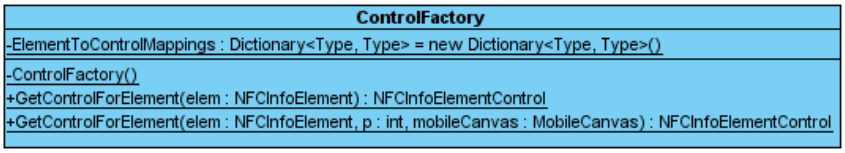


Figura 29 – *Factory* criada para obter um controlo para um dado elemento

No seu construtor de tipo, esta classe auxiliar obtém todas as classes que estendem `NFCInfoElementControl` (todos os controlos) e obtém o valor do atributo identificado anteriormente. O mapeamento criado nesta fase é utilizado depois no método `GetControlForElement` para decidir qual o tipo de controlo a instanciar, dado o tipo do elemento passado como argumento.

5.4.3 Componentes de edição de documentos

Os componentes de edição de documentos são aqueles que permitem a alteração do conteúdo ou do aspecto dos documentos. Nas secções seguintes são apresentados os três tipos de componentes criados para o efeito.

5.4.3.1 Inserção de novos elementos de informação

Estes componentes permitem adicionar novos elementos de informação. Estão agrupados numa janela, facilitando a sua escolha. Quando um dos botões contidos na janela é seleccionado, é adicionado o elemento respectivo ao documento. O elemento adicionado terá todas as propriedades com os valores por omissão. Esses valores podem depois ser alterados através da utilização dos componentes de edição de propriedades.
A Figura 30 apresenta os tipos de elementos actualmente disponíveis.

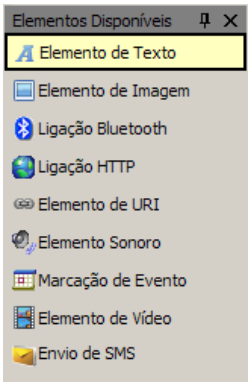


Figura 30 – Janela de inserção de novos elementos

5.4.3.2 Edição de propriedades dos elementos actuais

Esta funcionalidade é suportada através da utilização de um controlo disponibilizado na *Application Programming Interface* (API) *Windows Forms*, denominado `PropertyGrid`. Este controlo permite a edição de todas as propriedades públicas de um controlo.

Se determinado tipo não for suportado directamente pelo controlo, é possível adicionar suporte para esse tipo através da criação de um editor de tipo, ou de um conversor de tipo. Este é o caso da edição do identificador de conteúdo nos elementos de ligação. O identificador de conteúdo é uma instância do tipo `System.Guid`, que identifica qual o conteúdo que se deseja aceder utilizando determinado elemento de ligação. Sem a utilização de um editor de tipo, a introdução deste identificador não é trivial, e exige que o utilizador escreva (ou copie/cole) o identificador desejado. Para facilitar este processo foi criado um editor de tipo, apresentado na Figura 32.

Figura 31 – Apresentação do valor definido com a ajuda do editor de tipo

Este editor de tipo permite pesquisar pontos de informação remotos, e de seguida seleccionar o desejado. Tal como referido anteriormente, um ponto de informação possui associado um documento.

Na Figura 31 é apresentada a instância do controlo do tipo `PropertyGrid` que é utilizada para editar os valores das propriedades. Nesta figura, é possível identificar a propriedade que define o identificador do conteúdo remoto, bem como o seu valor, após a utilização do editor de tipo. O botão com reticências que surge à direita do valor actual da propriedade indica que esta propriedade tem um editor de tipo associado, que será invocado quando este botão for seleccionado.

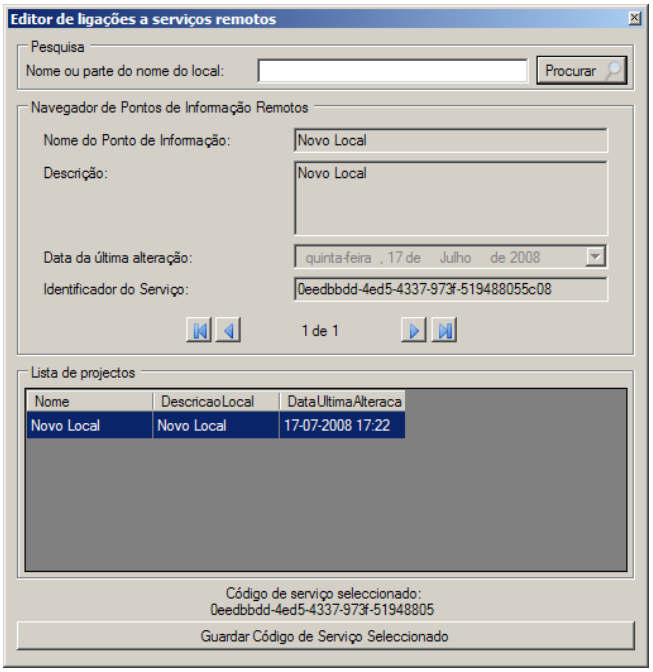


Figura 32 – Editor de tipo para identificadores de conteúdos remotos

Normalmente, o editor de tipo é associado ao tipo em questão. Caso o tipo não seja definido pelo programador, é possível associar o editor de tipo na propriedade. Esta solução é apresentada no Exemplo 7.

Exemplo 7 – Associação de um editor de tipo a uma propriedade

```
(...Outros atributos...)
[Editor(typeof(AssociatedRemoteServiceEditor), typeof(UITypeEditor))]
public Guid ServiceName
{
  get { return serviceName; }
  set { serviceName = value; OnPropertyChanged(); }
}
```

O editor de tipo pode ter o aspecto de uma janela, como era o caso do editor de tipo apresentado na Figura 32, ou o aspecto de uma *drop down list*. A forma de apresentação é decidida pelo editor de tipo e, tipicamente, depende do espaço necessário para a edição do valor. A Figura 33 apresenta outro editor de tipo criado durante o desenvolvimento da aplicação de edição de conteúdos.

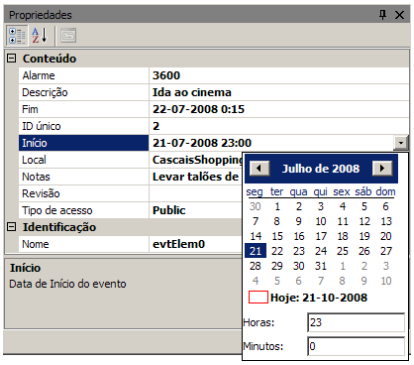


Figura 33 – Editor de tipo para data e hora

O editor de tipo ilustrado permite a escolha da data e hora de forma visual, não necessitando de uma janela para a edição do valor da propriedade.

Foram criados vários editores de tipo durante o desenvolvimento deste projecto.

Por vezes surge outra necessidade relacionada com a edição de propriedades, como é o caso da apresentação de valores booleanos, e que se prende com a necessidade de converter o valor real da propriedade em texto que seja mais amigável para o utilizador, e vice-versa. Este comportamento é conseguido através da utilização de conversores de tipo. Se não for especificado um conversor de tipo, a `PropertyGrid` apresenta o valor de uma propriedade do tipo `System.Boolean` com os valores "True" e "False". Para tornar esta apresentação mais natural, foi criado um conversor de tipo que converte os valores "True" e "False" em "Sim" e "Não" respectivamente. A Figura 34 apresenta o resultado da aplicação de um conversor de tipo a uma variável do tipo `System.Boolean`.

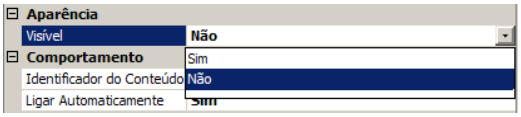


Figura 34 – Utilização de conversores de tipo

A associação do conversor de tipo à propriedade ou tipo é realizada da mesma forma que foi apresentada para os editores de tipo.

5.4.3.3 Edição de informação multimédia

Um caso especial em que o modelo anterior não pode ser utilizado é para a edição de conteúdos multimédia, uma vez que o conteúdo multimédia necessita de ser adicionado de forma diferente. As razões desta diferença estão relacionadas com limitações da plataforma *Java* ME, apresentadas na Secção 5.6.8.1. Assim, pontos de informação multimédia não são editáveis da mesma forma que os restantes pontos de informação (locais e remotos). Para estes casos foi criada a interface de edição de conteúdos multimédia apresentada na Figura 35.

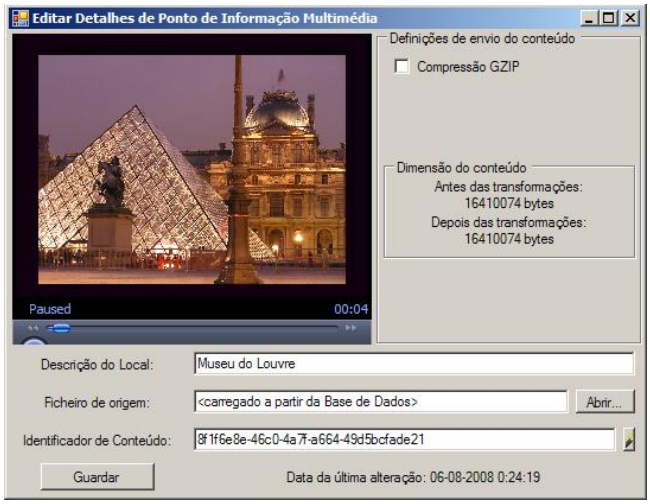


Figura 35 – Editor de elementos de informação multimédia

Esta interface permite a escolha e gravação de conteúdos multimédia a partir de ficheiro. Através da utilização de uma instância do tipo `AxMediaPlayerControl`, permite a pré-visualização dos conteúdos escolhidos. Este controlo é um *wrapper* na plataforma .NET para o controlo *ActiveX* `MediaPlayerControl`.

Posteriormente, a associação de conteúdo multimédia a elementos multimédia é feita seguindo o mesmo processo tal como foi apresentado para a associação de conteúdos a elementos de ligação, com a utilização de um editor de tipo para facilitar a introdução do valor do tipo `System.Guid` que identifica o conteúdo multimédia.

5.4.3.4 Manipulação e organização dos elementos dentro do documento actual

Os elementos podem ser reorganizados através da utilização dos controlos de movimentação para cima e para baixo, identificados pelas setas a verde, no canto inferior direito da Figura 36. É também possível eliminar o elemento actualmente seleccionado.

Os restantes botões que se encontram no canto superior direito da Figura 36 permitem utilizar a área de transferência (*clipboard*) para realizar as operações copiar, colar e cortar. Este foi um processo para o qual também foi utilizado o mecanismo de seriação descrito anteriormente.

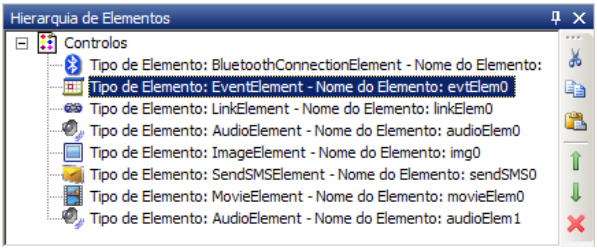


Figura 36 – Componentes para manipulação e organização dos elementos

A área de transferência disponibiliza um local para armazenamento de informação. É possível armazenar a informação na forma de sequências de bytes. Assim, quando é feita a cópia do elemento para a área de transferência, o elemento é seriado. Quando é lido de volta (ou seja, quando é feita a colagem), é deseriado para o seu estado original.

5.4.4 Componentes de escrita ou envio de conteúdos

Estes componentes permitem a escrita de conteúdos em *tags Mifare* ou em ficheiro. A escrita é efectuada após a conversão da representação em memória do documento numa sequência de bytes. Esta conversão é efectuada utilizando o motor de seriação de conteúdos.

5.4.4.1 *Tags* Mifare

A escrita de *tags Mifare* é efectuada utilizando a interface apresentada na Figura 37. Esta interface permite definir o número de *tags* que se deseja escrever. Após a escrita de uma *tag*, a interface apresenta uma mensagem na qual é pedida a colocação de uma nova *tag*. Quando é detectada a presença de outra *tag*, o processo de cópia continua até ser atingido o número de cópias pretendido.

As *tags* são gravadas para que, quando forem lidas por um dispositivo móvel que ainda não tem a aplicação instalada, seja desencadeado o processo de *download* da mesma. Para esse efeito, é possível definir o endereço a partir do qual é possível efectuar a instalação da aplicação. Serão apresentados mais detalhes deste processo na Secção 5.6.5.2.

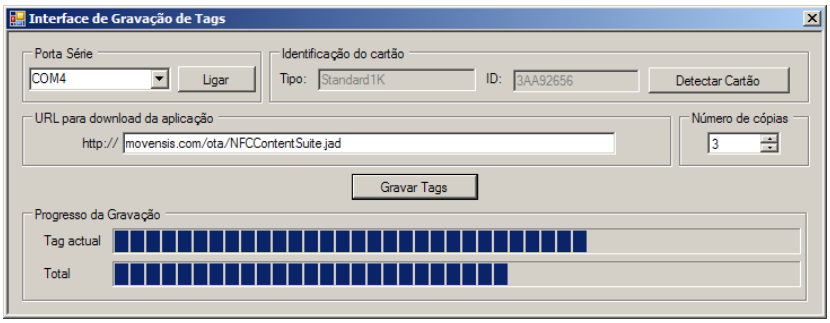


Figura 37 – Interface de gravação de *tags* Mifare

5.4.4.2 Ficheiro

Este componente permite a escrita de um ficheiro com o conteúdo seleccionado. Isto pode ser útil caso seja necessário fornecer conteúdos num terminal que não tenha acesso ao serviço de dados, mas que suporte o envio de informação através de NFC-IP, *Bluetooth* ou outro tipo de ligação suportada. Desta forma, o conteúdo seria obtido a partir do ficheiro e enviado directamente para os terminais.

5.4.5 Componentes para facilitação do processo de edição de conteúdos

Para facilitar o processo de criação de conteúdos, foi criada a possibilidade de efectuar a pré-visualização nos dispositivos móveis dos conteúdos que se encontram a ser editados na aplicação de edição. Isto evita a necessidade da escrita de *tags* para validar se o conteúdo desenhado é exibido da forma esperada na aplicação de exibição de conteúdos.

5.4.5.1 NFC-IP

Este componente permite o envio da informação necessária para exibir o conteúdo que está a ser editado na aplicação de edição de conteúdos através da utilização do protocolo NFC-IP1. A aplicação de exibição de conteúdos (*Java* ME) exibe essa mesma informação, utilizando o mesmo mecanismo que será posteriormente utilizado quando a aplicação estiver em funcionamento normal.

5.4.5.2 Bluetooth

Este componente é semelhante ao que foi identificado no ponto anterior, com a diferença do protocolo de comunicação utilizado que neste caso é *Bluetooth*. Para suportar este modo, foi criada uma versão modificada da aplicação de exibição de conteúdos. Este módulo de entrega de conteúdos por *Bluetooth* publica um serviço que é depois procurado pela versão modificada da aplicação de exibição de conteúdos.

5.5 Bibliotecas .NET para suporte à tecnologia NFC

Durante a realização deste projecto, foi identificada a necessidade de criar componentes relacionados com a tecnologia NFC. Alguns destes componentes estão relacionados com a estrutura dos dados e outros com a comunicação com o *hardware*. Estes componentes foram criados de forma a serem genéricos, para possibilitar a sua utilização em vários projectos.

5.5.1 Biblioteca de comunicação com leitores externos

São requisitos deste projecto a composição e gravação de conteúdos em *tags* NFC para posterior consulta no terminal móvel, bem como a possibilidade da troca de conteúdos directamente entre o dispositivo móvel e um computador.

5.5.1.1 Apresentação técnica do leitor NFC utilizado

Para a implementação desta biblioteca, foi utilizado um dos poucos leitores/escritores NFC disponíveis ao público à data do arranque deste projecto, denominado *NFC/Mifare Desktop Reader APPA 13,56MHz*, do fabricante Arygon Technologies.

Este leitor encontra-se representado na Figura 38 e inclui os seguintes componentes[22]:

- Interface *Universal Serial Bus* (USB) que simula uma porta série
- Micro controlador proprietário (ADRA)[22] que disponibiliza um conjunto de comandos de alto nível para manipulação de *tags Mifare*.
- Micro controlador do fabricante NXP (PN531)[23] que inclui toda a lógica relacionada com o protocolo NFC-IP1 e com o acesso às *tags* NFC e *Mifare Standard*
- Antena
- Dois *Light Emitting Diodes* (LED), com as cores verde e vermelho, para apresentação de feedback do estado do leitor de forma visual.

Figura 38 – Aspecto do leitor NFC utilizado

A organização interna do leitor em causa é apresentada na Figura 39.

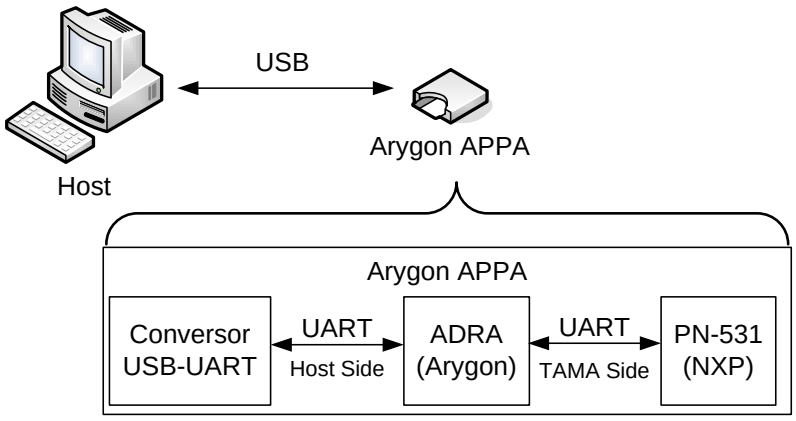


Figura 39 – Detalhe da estrutura do leitor Arygon APPA

5.5.1.2 Implementação da componente de acesso a *tags* Mifare

Durante a fase de implementação, foi considerada a aquisição de um leitor do fabricante Kimaldi, tendo sido utilizada uma unidade de avaliação. No entanto, já existia também o contacto com o fabricante Arygon, pelo que a implementação foi influenciada por estas condicionantes. Para minimizar este problema, foi definida uma interface que todos os leitores tipicamente implementam (`IMifareReaderOperations`), e no final, acabaram por ser criadas três implementações: uma que interage com o leitor do fabricante Arygon descrito anteriormente, outra que interage com o leitor do fabricante Kimaldi, e outra que gere um leitor virtual. Esta interface é apresentada na Figura 40.

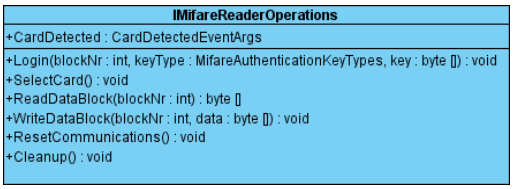


Figura 40 – Interface que define o conjunto mínimo de operações que um leitor/escritor de *tags* Mifare tem de suportar

A implementação do leitor virtual simula a presença de uma *tag* Mifare de 1K (MF1CS50)[13]. É possível manipular a memória deste leitor tal como seria de esperar de um leitor físico. Esta implementação foi utilizada para a fase de testes inicial enquanto não foi possível interagir com o dispositivo real. O leitor virtual simula apenas o acesso à memória, e não implementa o modelo de autenticação *Mifare*.

As aplicações podem optar por instanciar directamente uma das implementações desta interface para obter uma instância que implemente esta interface. Isto obriga que seja escolhido o tipo de leitor a ser utilizado em tempo de compilação. Para permitir esta escolha através de ficheiros de configuração, foi criado um modelo de *providers*.[24] Cada implementação dos leitores é um *provider*. Neste modelo a instanciação do tipo que representa o leitor é feita através da consulta do ficheiro de configuração, onde é identificado qual o *provider* que está em uso. Após obtido o tipo, é criada a instância a ser usada pela aplicação, que não necessita de conhecer qual o tipo específico de leitor com o qual está a comunicar. Este processo de obtenção da instância adequada é facilitado através da utilização do padrão de desenho *factory*[21]. Desta forma, o tipo de leitor que será utilizado nas aplicações pode ser definido através do ficheiro de configuração. É também possível adicionar suporte a outros tipos de leitor sem ser necessário recompilar a aplicação. A implementação destes novos leitores pode residir num *assembly* que é carregado dinamicamente através da informação presente no ficheiro de configuração. O Exemplo 8 ilustra o ficheiro de configuração para o caso do leitor virtual apresentado.

Exemplo 8 – Ficheiro de configuração do modelo de providers de leitores *Mifare*

```
<?xml version="1.0" encoding="utf-8" ?>
<providermodel defaultname="Virtual Reader">
  <provider name="Virtual Reader"
            type="NFCInfo.RFID.Readers.Virtual.VirtualMifareReaderProvider,
NFCInfo.RFID.Readers.Virtual, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
            description="Virtual Mifare Reader Provider"
            parameters=""/>
</providermodel>
```

A classe `ArygonAPPAReader`, que implementa a interface `IMifareReaderOperations` para o leitor da Arygon, utiliza tipos auxiliares que foram desenhados numa arquitectura de camadas (ilustrada na Figura 41), de forma a criar diferentes níveis de abstracção do hardware.

O micro controlador ADRA da Arygon (que faz parte do leitor apresentado anteriormente) disponibiliza um protocolo de alto-nível para acesso às funcionalidades *Mifare*. Este modo cria uma interface mais simples para acesso às funcionalidades relacionadas com a família de *tags Mifare*. Neste modo, os comandos seguem o protocolo definido para este micro controlador. No entanto, também é possível obter as mesmas funcionalidades comunicando apenas com o micro controlador PN531. Esta abordagem é necessária para o caso do protocolo NFC-IP1, como será detalhado na secção 5.5.1.4, uma vez que o micro controlador ADRA não suporta este protocolo.

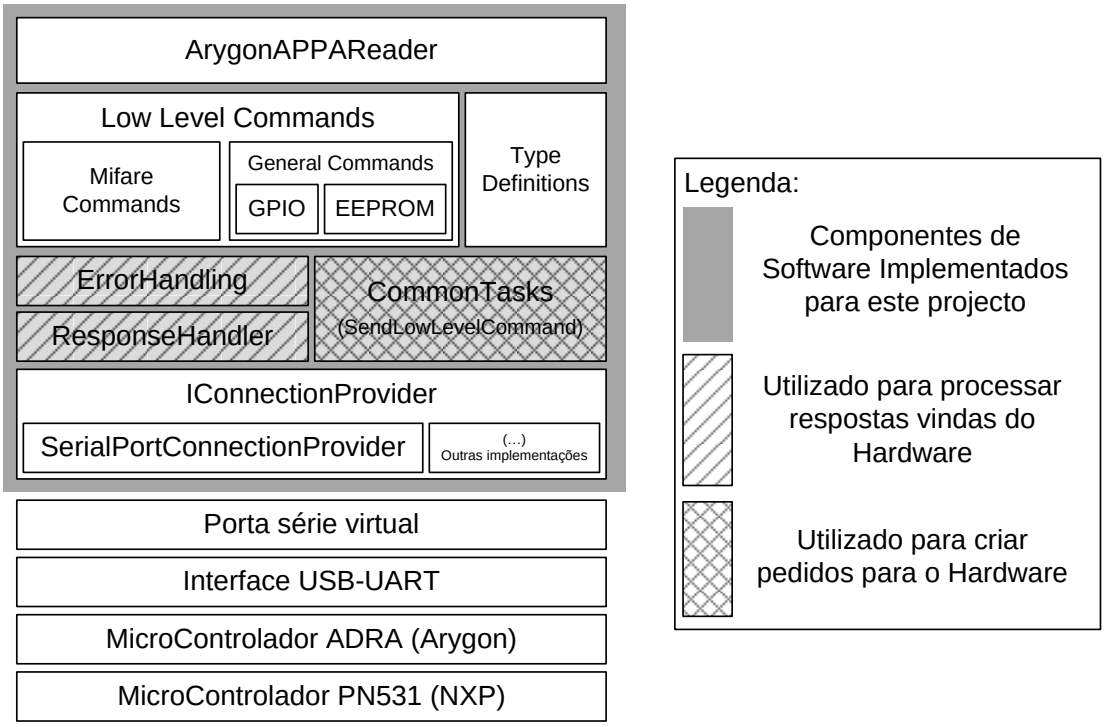


Figura 41 – Esquema das camadas de software e hardware relacionadas com o leitor ADRA (Arygon)

Em termos de *software*, o nível inferior foi desenhado de forma a ser independente do meio de comunicação utilizado. Assim, foi criado o conceito de fornecedor de ligação: foram definidas as operações que cada fornecedor deve suportar, e foi realizada uma implementação deste conceito para o caso da porta série. Se mais tarde a interface do leitor for alterada, este será o único ponto que necessitará de ser refeito. No Exemplo 9, é apresentada a utilização deste modelo no módulo de envio de comandos.

O nível imediatamente superior está dividido em três componentes. Dois deles processam as respostas vindas do hardware, e o restante suporta a criação de comandos e desencadeia o respectivo processamento (envio, espera da resposta e encaminhamento para os componentes de tratamento da resposta). Este nível cria uma abstracção do protocolo, que facilita a implementação dos níveis superiores, que necessitam apenas de identificar o comando a executar.

Exemplo 9 – Processamento de comando e utilização de `ConnectionProviders`

```
public static String SendLowLevelCommand(String cmd)
{
  //Enviar identificador de modo (ASCII = 0)
  ConnectionProviderManager.CurrentConnectionProvider.WriteString("0");
  //Enviar o comando (Modo ASCII)
  ConnectionProviderManager.CurrentConnectionProvider.WriteString(cmd);
  //Esperar pela resposta e processá-la
  return ResponseHandler.HandleResponse();
}
```

Caso o processamento do comando resulte em erro, é lançada uma excepção com a mensagem adequada ao erro detectado. Caso contrário, são removidos os dados de controlo do protocolo (indicação de erros, CRC, entre outros) e é devolvido o resultado efectivo da operação. Este resultado depende do comando em causa, e é interpretado pela camada superior (`LowLevelCommands`), como no Exemplo 10. Aqui, após a interpretação do resultado da operação, é devolvida para a camada superior a informação extraída, utilizando instâncias de tipos definidos para o efeito.

Exemplo 10 – Utilização dos serviços das camadas inferiores, na camada `LowLevelCommands`, retornando uma instância de um tipo composto

```
public static CardInfo SelectCard()
{
  //Pedir que seja seleccionada uma tag..
  CommonTasks.SendLowLevelCommand("s");
  //Aguardar que seja seleccionada uma tag, e obter dados
  String responseBuffer = ResponseHandler.HandleResponse();
  CardInfo cardInfo = new CardInfo();
  //Preenchimento da instância do tipo CardInfo, usando a resposta obtida
  (...)
  return cardInfo;
}
```

A classe que é depois utilizada nas aplicações para interacção com o leitor (`ArygonAPPAReader`) mantém informação de estado acerca do leitor, e implementa a interface `IMifareReaderOperations` definida anteriormente. No Exemplo 11 é apresentada a utilização nesta classe dos serviços das camadas inferiores apresentados anteriormente.

Exemplo 11 – Utilização dos serviços das camadas inferiores, na classe *ArygonAPPAReader*

```
public bool GreenLedState
{
  set
  {
    if (!ledsInitialized)
    {
      MiscCommands.InitLeds();
      ledsInitialized = true;
    }
    MiscCommands.SetLedState(LedColors.Green, value ? LedStates.On : LedStates.Off);
  }
}
```

5.5.1.3 Interface de testes para a componente de escrita de *tags Mifare*

Durante a fase de testes das bibliotecas de comunicação com leitores externos e escrita de *tags Mifare*, foi criada uma aplicação de testes que para além do teste da comunicação com o dispositivo físico, permitisse a

visualização e manipulação dos dados presentes em *tags Mifare*. O aspecto desta aplicação é apresentado na Figura 42.

Esta aplicação permite a visualização e edição do conteúdo da *tag*, removendo os campos de controlo. Esta organização foi escolhida porque esta aplicação foi muito utilizada também durante o desenvolvimento da implementação da especificação JSR-257[25], para comparar as *tags* escritas pela aplicação de edição de conteúdos, com as *tags* escritas pelo dispositivo móvel.

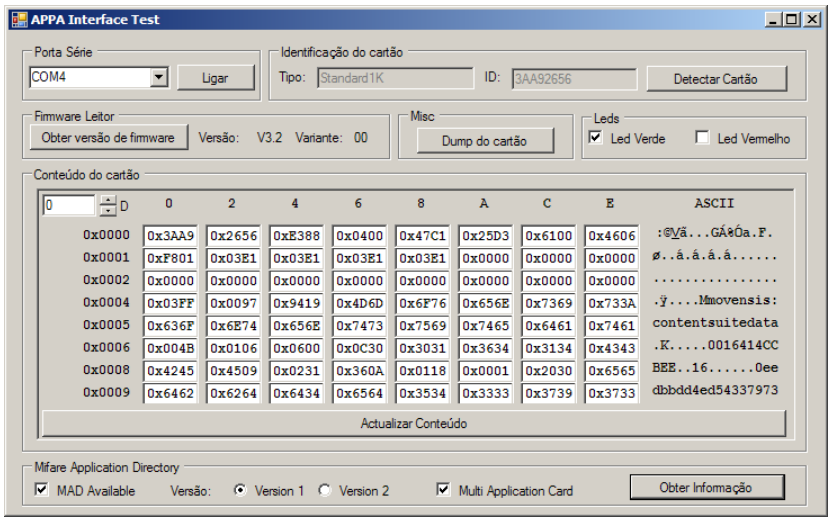


Figura 42 – Interface de testes para a componente de escrita de *tags Mifare*

Esta aplicação foi desenvolvida utilizando o padrão de desenho *Model-View-Controller*[26].[21] Para esse efeito, foi definida uma interface que indica a funcionalidade que a *View* deve suportar (`IArygonAPPAView`). De seguida, foi criada a *Form* (classe `ArygonAPPA`) que implementa essa interface. Esta classe traduz os eventos efectuados sobre controlos gráficos em eventos da *View*. Quando a aplicação se inicia, esta classe é registada como *View* no *Controller* actual. A partir deste registo, a classe `ArygonAPPAController` passa a receber os eventos gerados pela *View*, e quando são efectuadas alterações no *Model*, actualiza a *View* para reflectir essas mudanças. Neste caso, o *Model* é representado pela classe que implementa a comunicação com o leitor, que é manipulada pelo *Controller*.

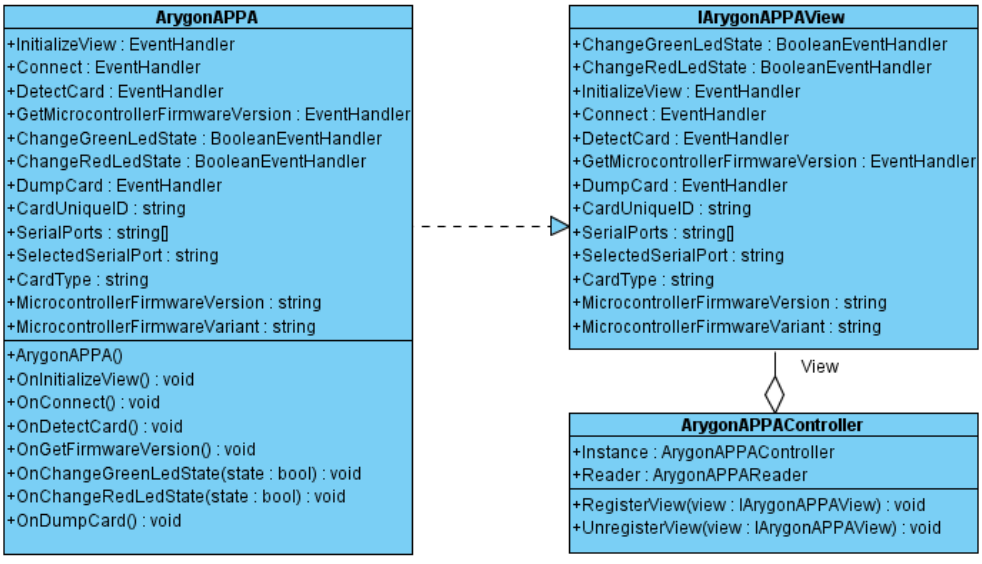


Figura 43 – Diagrama de classes da aplicação de testes para a componente de escrita de *tags Mifare*

Este padrão de desenho também foi utilizado na interface de gravação de *tags Mifare* apresentada anteriormente.

5.5.1.4 Implementação do suporte para o protocolo NFC-IP1

A implementação do suporte para o protocolo NFC-IP1 segue a mesma linha estrutural da implementação das funcionalidades *Mifare* descritas anteriormente. No entanto, uma vez que neste caso o micro controlador ADRA não oferece um protocolo em modo ASCII semelhante ao que foi descrito para o caso das funcionalidades *Mifare*, foi necessário criar uma estrutura análoga, mas que suporte o protocolo binário TAMA[1], utilizado pelo micro controlador PN-531. Neste caso, o micro controlador ADRA (Arygon) limita-se a encaminhar os comandos e respostas que recebe para o PN531 e para o *host* respectivamente. Neste caso, o *host* é um computador pessoal.

Para este projecto, apenas foi necessária a implementação da funcionalidade de envio de informação a partir do computador pessoal para um dispositivo que suporte NFC-IP. A recepção não foi implementada.

Para facilitar a utilização da funcionalidade de envio de informação através do leitor Arygon, utilizando o protocolo NFC-IP1, foi criado um *stream* de escrita (`NFCIP1OutputStream`). Esta classe fornece abstracção dos detalhes relacionados com o protocolo NFC-IP1, tais como detecção e selecção de dispositivos, limitações ao nível do tamanho de *buffers* e gestão de erros. Esta gestão será detalhada na Secção 5.6.6. Este *stream* permite a criação de aplicações de distribuição de conteúdos utilizando NFC-IP1, sem a necessidade do conhecimento dos detalhes associados a este protocolo, como ilustrado no Exemplo 12.

O *stream* criado é utilizado na aplicação de edição de conteúdos para fornecer a funcionalidade de pré-visualização utilizando NFC-IP1, apresentado na Figura 44.

Exemplo 12 – Utilização da classe *NFCIP1OutputStream* para o envio de conteúdos

```
using (NFCIP1OutputStream _nfcIPOutputStream = new NFCIP1OutputStream(_reader))
{
  byte[] data = (...) //Obter dados a enviar (utilizando motor de seriação)
  _nfcIPOutputStream.Write(data, 0, data.Length);
}
```

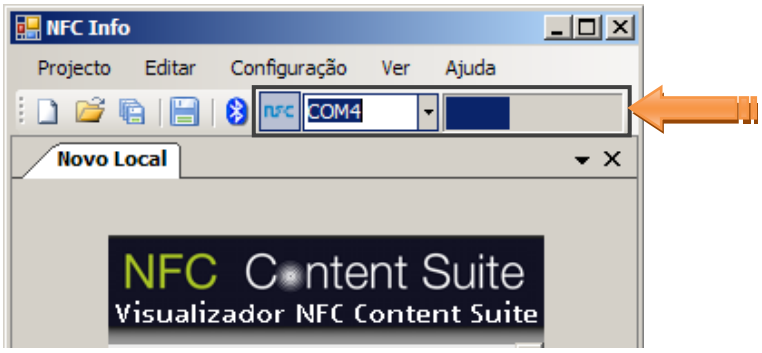


Figura 44 – Função de pré-visualização de conteúdos com envio através de NFC-IP1

[1] "TAMA" é também o nome de código do micro controlador PN-531

5.5.2 Implementação da especificação JSR-257

De forma a ser possível criar mensagens NDEF, contendo vários tipos de *Records* e obter a respectiva representação binária de acordo com as especificações do *NFC Forum*, foi criada uma implementação da especificação JSR-257 [25], utilizando a linguagem *C#*.

Apesar de não ter sido uma funcionalidade necessária para este projecto, esta biblioteca permite também a operação inversa ou seja, a criação de objectos a partir de sequências de bytes. Esta funcionalidade foi adicionada de forma a criar um componente independente para utilização noutras aplicações.

A implementação foi baseada na implementação e documentação de referência da especificação JSR-257, tendo sido feitas as adaptações necessárias para que o código final ficasse de acordo com as convenções de nomes normalmente utilizadas em *C#*. A Figura 45 apresenta o diagrama de classes implementado.

Figura 45 – Diagrama de classes da implementação da especificação JSR-257

Após a obtenção da representação binária da mensagem esta pode, por exemplo, ser escrita numa *tag*. Para além desta representação, no caso das *tags* Mifare, é ainda necessário criar uma *Application* no *Mifare Application Directory* (MAD)[27] com o identificador correcto.[28] O MAD é utilizado para associar sectores da *tag* à aplicação, identificada através de identificador atribuído pela NXP (fabricante da tecnologia *Mifare*). A utilização do MAD permite que várias aplicações partilhem a mesma *tag* sem se comprometerem mutuamente. Uma aplicação reserva determinado número de sectores para uso próprio, deixando os restantes disponíveis para outras aplicações.

Este componente foi identificado como sendo um subproduto deste projecto, uma vez que pode ser utilizado em qualquer outro projecto em que seja necessário proceder à manipulação de mensagens NDEF.

Após a escrita do conteúdo nas *tags*, é possível efectuar a sua leitura na aplicação *Java* ME. Esta última é responsável pela exibição do conteúdo no dispositivo móvel.

5.6 Aplicação *Java* ME para visualização dos conteúdos em dispositivos móveis

Para possibilitar tipos de conteúdo para além dos suportados directamente pelo *firmware* do dispositivo móvel, foi criada uma aplicação *Java* ME. A aplicação também é essencial para suportar o armazenamento e posterior envio de conteúdos. Esta aplicação obtém o conteúdo a partir de uma das fontes de informação disponíveis, cria uma representação em memória do documento e exibe-o no dispositivo do utilizador.

A representação em memória é criada utilizando os mesmos conceitos que foram enunciados para o motor de seriação apresentado na Secção 3.4.1. No entanto, para esta aplicação basta a implementação parcial deste motor (a componente de deseriação), uma vez que nesta aplicação não são gerados conteúdos.

Actualmente são suportadas as seguintes fontes de informação: *Bluetooth*, servidor HTTP, sistema de ficheiros, NFC-IP e *tags* NFC. No entanto, qualquer fonte de dados que possa ser modelada em forma de *stream* de leitura pode ser utilizada, uma vez que o motor de deseriação implementado está desacoplado do tipo de *stream* de leitura associado.

5.6.1 Plataforma *Java* ME

A aplicação de visualização de conteúdos foi implementada utilizando a plataforma *Java ME*[29]. A escolha desta plataforma deve-se ao facto de esta ser a plataforma na qual é possível criar aplicações utilizando a tecnologia NFC, no protótipo escolhido para a realização deste projecto.

5.6.2 Apresentação técnica do protótipo utilizado

O desenvolvimento deste projecto foi possível devido à cedência, por parte da subsidiária portuguesa da Nokia, de protótipos do único dispositivo lançado pela empresa à data de arranque deste projecto, o modelo 6131 NFC, cuja arquitectura se apresenta na Figura 46.

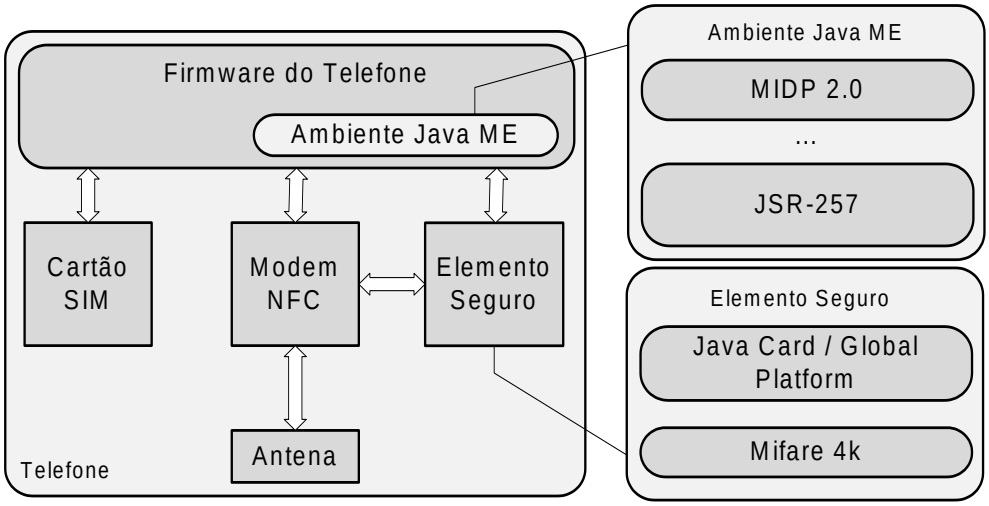


Figura 46 – Arquitectura do terminal utilizado para o desenvolvimento

Este terminal possui um sistema operativo proprietário, que inclui o ambiente *Java ME*. Este ambiente é baseado na configuração CLDC 1.1 e no perfil MIDP 2.0. Para além disto, contém ainda vários *packages* opcionais e, entre eles, os que dão suporte à utilização das várias vertentes da tecnologia NFC a partir do ambiente *Java ME*. [30]

O componente identificado como Elemento Seguro, é uma parte muito importante para aplicações que necessitem de armazenamento seguro, por exemplo, para armazenar bilhetes ou créditos. No entanto, para este projecto não faz sentido a sua utilização.

Neste dispositivo, por decisões de desenho, ou restrições de dimensão do mesmo, a antena é de uma dimensão inferior à normalmente utilizada nos leitores e nas *tags*. Isto resulta num alcance reduzido, e exige que os dispositivos que desejem trocar informação com este terminal se encontrem muito perto (distâncias na ordem dos 0-2 cm). No caso da comunicação através de NFC-IP1, em que existem dois dispositivos com as mesmas limitações em termos de dimensão da antena, a comunicação só se realiza quando os dispositivos estão de facto encostados. Não é claro se esta é uma limitação que não foi resolvida, ou se é resultado de decisão relacionada com segurança.

Esta limitação é por vezes apresentada como trazendo segurança adicional, uma vez que é necessário que os terminais estejam realmente perto para que se desencadeie a troca de informação. O aspecto deste dispositivo e a localização da antena NFC são apresentados na Figura 47.

Figura 47 – Aspecto do terminal utilizado, e indicação do local da antena NFC

Este dispositivo suporta três modos de operação distintos. A forma como estes modos se relacionam com outros tipos de *hardware* é apresentada na Figura 48.

1. *Reader/Writer*
    a. Neste caso o dispositivo comporta-se como um leitor/escritor de *tags* e *Smart Cards*. Em conjunto com a utilização de mensagens NDEF, este modo permite a criação e leitura de conteúdos simples de *tags* NFC.
2. Emulação de *tag* ou *Smart Card*
    a. Neste modo o dispositivo simula a existência de um *Smart Card* ISO-14443 ou de uma *tag Mifare* 4k[31]. Quando este modo está activo, o dispositivo funciona como se se tratasse de uma *tag*. Este funcionamento é inteiramente suportado pelo dispositivo, evitando a necessidade da alteração das infra-estruturas de leitura de cartões sem contacto existentes.

3. *Peer-to-Peer*
    a. Este modo permite a troca de informação directamente entre dois dispositivos, utilizando o protocolo NFC-IP1[6][32]. Neste dispositivo, este modo é apenas suportado pelas aplicações *Java* ME. O sistema operativo do dispositivo não permite o envio de conteúdos por este meio.

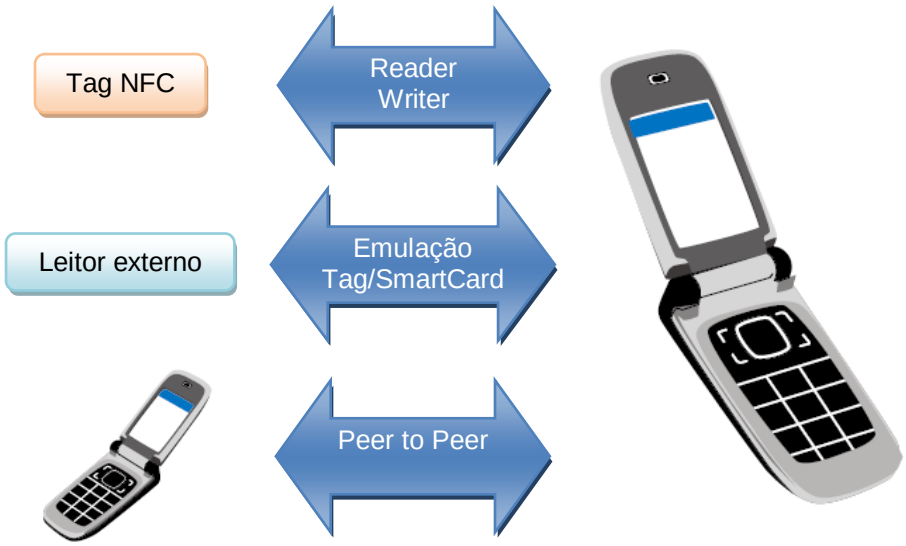


Figura 48 – Relação entre os diferentes modos de funcionamento e restantes dispositivos NFC

O modem NFC permite a utilização de apenas um modo em determinado instante. No entanto, o *firmware* do dispositivo alterna entre os diferentes modos de utilização de forma automática, simulando o funcionamento em mais do que um modo em simultâneo.

O *firmware* do dispositivo suporta o tratamento de mensagens NDEF que contenham os seguintes tipos de *records*:

Nome do formato	Nome do tipo	Descrição
MIME	text/x-vCard	Cartão de visita
MIME	text/x-vCalendar	Nota de calendário
RTD	urn:nfc:wkt:Sp	*Record Smart Poster*
RTD	urn:nfc:wkt:U	*Record* URI
External RTD	urn:nfc:ext:nokia.com:bt	*Record* com dados para emparelhamento *Bluetooth*

Tabela 1 – Tipos de dados suportados pelo *firmware* do dispositivo

Adicionalmente, os tipos identificados a sombreado na Tabela 1 são sempre tratados pelo *firmware* do dispositivo, e nunca são passados às aplicações.

O *NFC Forum* classificou as tags NFC em quatro tipos. As *tags* classificadas segundo estes tipos variam em características tais como tamanho, controlo de acesso à informação, família e organização da memória.

Este dispositivo suporta todos os quatro tipos de *tag* definidos pelo *NFC Forum* (descritos no anexo 7.1), e adicionalmente, apesar de não estarem classificados em qualquer um dos tipos identificados anteriormente, são ainda suportados os seguintes tipos de *tags*:

- *Mifare Standard 1k*.
- *Mifare Standard 4k*.
- *Innovision Jewel*.

Estes tipos são suportados devido ao facto de serem tipos de *tag* muito usados no mercado.

Este dispositivo utiliza internamente a mesma versão do circuito integrado da NXP descrito anteriormente (PN531)[23][33], que é utilizado no leitor/escritor NFC da Arygon.

5.6.3 Arranque automático da aplicação

Um ponto forte da tecnologia NFC é o facto de esta ser de utilização muito intuitiva. Tipicamente, o utilizador apenas necessita de encostar dois dispositivos NFC para que se desencadeie uma acção. Este comportamento é também desejável na aplicação de visualização de conteúdos, para que arranque automaticamente e exiba o conteúdo quando o terminal ficar em contacto com outro dispositivo NFC.

Este comportamento é conseguido recorrendo ao suporte existente em todos os dispositivos que incluam a plataforma *Java* e o perfil MIDP 2.0. O nome do suporte em causa é *push registry*, e permite que aplicações que não se encontram em execução sejam iniciadas em resposta a um evento externo tal como a recepção de mensagens de SMS, recepção de dados via *sockets*, entre outros.

Esta funcionalidade é suportada por um componente específico de cada plataforma ou dispositivo, denominado AMS (*Application Management Software*). Na presença de um destes eventos externos, o AMS verifica se existem aplicações registadas para esse evento, e em caso afirmativo provoca a sua execução.

A associação a um destes eventos pode ser feita de forma programática, utilizando os métodos disponíveis na classe `PushRegistry` do *package* `javax.io` (denominado de registo dinâmico), ou através de atributos do tipo MIDlet-Push-[número], adicionados ao ficheiro JAD(*Java Application Descriptor*) que descreve a aplicação *Java* ME (registo estático).

Independentemente da forma como é feito o registo, são sempre indicados três argumentos aquando da instalação:

1. A sequência de caracteres que identifica a ligação (exemplo: socket://:7000).
2. O nome da *midlet* a ser colocada em execução[1]
3. Um filtro com conteúdo dependente do tipo de ligação, que permite que a mesma ligação seja utilizada para arranque automático de várias aplicações (exemplo: endereço IP de origem, no caso das *sockets*).

[1] O nome é necessário porque o ficheiro JAD é utilizado para descrever *MIDlet Suites*, que por sua vez podem conter uma ou mais *MIDlets*.

A especificação JSR-257 adiciona suporte para outro tipo de evento externo: o contacto com *tags* NFC. Neste tipo de eventos externos, a ligação é identificada por um URL NDEF, que segue a seguinte estrutura[25]:

<ndef url> ::= "ndef:"<record_type_format>?name=<record_type_string>

<record_type_format> ::= "rtd" | "external_rtd" | "mime"

<record_type_string> ::= Cadeia de caracteres US_ASCII, com o nome completo (*fully qualified name*) do RTD

De seguida, são apresentados alguns exemplos da utilização desta estrutura, e respectiva descrição

- **ndef:mime?name=image/jpeg**
  - Registo para arranque de uma aplicação quando é lido um *record* MIME, com informação do tipo "image/jpeg".
- **ndef:rtd?name=urn:nfc:wkt:Sp**
  - Registo para arranque de uma aplicação quando é lido um *record* RTD do tipo Smart Poster.
- **ndef:external_rtd?name=urn:nfc:ext:movensis:contentsuitedata**
  - Registo para arranque de uma aplicação quando é lido um *record* do tipo *External RTD*, com o nome `movensis:contentsuitedata`.

Para o caso de eventos externos relacionados com *tags* NFC, o filtro da ligação pode ser utilizado para filtrar o arranque através do identificador da *tag* lida.

O Exemplo 13 apresenta o registo criado para o arranque automático da aplicação desenvolvida:

Exemplo 13 – Excerto do ficheiro JAD, exibindo o atributo responsável pelo registo no *push registry*

```
MIDlet-Push-1:
ndef:external_rtd?name=urn:nfc:ext:movensis:contentsuitedata,com.movensis.midlets.NFCCon
tentSuiteMidlet,*
```

Desta forma, a *midlet* `NFCContentSuiteMidlet` contida no *package* com.movensis.midlets será iniciada quando for lida uma mensagem NDEF com um *record* do tipo *External RTD*, com o nome `movensis:contentsuitedata`, a partir de qualquer *tag* NFC.

O tipo de mensagem e do *record* é dado pela cadeia de caracteres que identifica a ligação, enquanto o facto de não ser necessária uma *tag* em particular é dado pelo *wildcard* "*" utilizado no filtro.

5.6.4 Assinatura digital da aplicação

Durante o desenvolvimento da *midlet Java* ME, surgiu a necessidade de utilizar determinadas APIs do dispositivo que estão protegidas. Se a aplicação tentar aceder a recursos tais como ligações *Bluetooth* ou a ficheiros no cartão de memória, será pedida ao utilizador a permissão para que a aplicação aceda a tais recursos. Em certos casos, pode até suceder a falha da aplicação devido ao facto de que para certas permissões o utilizador não é questionado, e a chamada ao método protegido falha com o lançamento de uma excepção do tipo `java.lang.SecurityException`.

A solução para este problema passa pela assinatura da *midlet suite* utilizando a chave privada associada ao certificado digital emitido por uma CA (*Certificate Authority*) cujo certificado raiz esteja instalado no dispositivo móvel, de modo a mudar o domínio de protecção da *midlet suite*.

Todas as aplicações *Java ME* são associadas, após a instalação no dispositivo móvel, a um domínio de protecção. O domínio de protecção caracteriza o conjunto de permissões e de modos de interacção.
Estas permissões podem ser concedidas automaticamente ou negadas até que o utilizador as conceda explicitamente.

O modo de interacção define a forma como a permissão é obtida. Existem quatro modos de interacção para as permissões:

1. *Blanket*

   Neste modo, a permissão é sempre concedida, e o utilizador não é sequer notificado acerca da sua necessidade.

2. *Session*

   Quando é utilizado este modo, apenas é pedida uma vez a permissão por cada sessão de utilização da aplicação. Assim que a mesma for terminada e carregada novamente, a permissão será novamente requisitada.

3. *Oneshot*

   Neste modo, o utilizador concede permissão apenas para a chamada actual à API protegida. Este modo é especialmente nocivo para a experiência de utilização, se a permissão desejada for utilizada com alguma frequência na aplicação.

4. *No*

   Este modo de interacção indica que não é possível que esta permissão venha a ser concedida, uma vez que o utilizador não será questionado acerca da sua aprovação. A permissão é negada, e a aplicação em causa recebe uma excepção do tipo `java.lang.SecurityException` a indicar o problema.

Todos os dispositivos que suportam a plataforma *Java ME* possuem um componente que é responsável pela instalação e gestão de aplicações, denominado AMS (referido na secção anterior). Este é responsável pela atribuição de determinado domínio de protecção às aplicações após a sua instalação. Este domínio é depois utilizado para determinar quais as permissões e respectivos modos de interacção da aplicação.

Cada domínio de protecção, excepto o domínio *Untrusted*, está associado a um conjunto de certificados raiz.

Quando a *midlet suite* está assinada, é necessário que seja utilizado um certificado que possa ser validado utilizando um desses certificados raiz.
É a partir desta correspondência que determinada aplicação é colocada no domínio de protecção adequado. Para evitar possíveis conflitos na associação de domínios de segurança a uma aplicação assinada, cada certificado raiz está associado a apenas um domínio de segurança.

A especificação MIDP 2.0 recomenda quatro domínios de protecção para dispositivos *Global System for Mobile communications* (GSM) ou *Universal Mobile Telecommunications System* (UMTS):

1. Fabricante

   Este domínio utiliza certificados raiz pertencentes ao fabricante do dispositivo. Tipicamente estes certificados são utilizados para assinar apenas as aplicações do fabricante, ou de terceiros em que o fabricante confie.

2. Operador

   Neste domínio de segurança são colocadas as aplicações assinadas por certificados que sejam verificáveis utilizando um certificado raiz gerado pelo operador de telecomunicações. Este certificado é tipicamente instalado no dispositivo antes da venda do mesmo.

3. *Trusted Third Party*

   É neste domínio que são colocadas as aplicações que são assinadas utilizando certificados adquiridos a uma CA, É também aqui que a aplicação desenvolvida no decorrer deste projecto é colocada, após a sua instalação. A aplicação encontra-se assinada com um certificado digital adquirido à empresa *Verisign*, que é uma das CAs cujo certificado raiz se encontra instalado no dispositivo NFC utilizado para o desenvolvimento da aplicação.

4. *Untrusted*

   Finalmente, para as aplicações que não se encontram assinadas, bem como para as aplicações MIDP 1.0 (que não suportam este modelo de segurança), existe este domínio de segurança. Tipicamente possui um conjunto de permissões, e respectivos modos de interacção mais restritos, para todas as aplicações que não requerem acesso a funcionalidades restritas do dispositivo.

Apesar de pertencer ao domínio *Trusted Third Party*, a aplicação continua a não ter o modo de interacção *blanket* activado por omissão em todas as permissões. No entanto, neste caso, se o utilizador confiar no fabricante da aplicação (que se encontra identificado através da assinatura digital), pode alterar o modo de interacção para as permissões que a aplicação necessita e evita a necessidade de confirmar constantemente a cedência de determinada permissão. Esta configuração é também suportada pelo AMS.

Estas configurações são controladas tendo em conta o domínio de protecção da aplicação. Por exemplo, numa aplicação que pertença ao domínio de protecção *Untrusted*, o modo de interacção para permissões de acesso aos dados privados do utilizador pode ser restringido pelo AMS de forma a não poder ser configurado como *Blanket*.

Uma aplicação *Java ME* é composta por dois ficheiros. Um ficheiro com extensão JAR (*Java ARchive*) que contém o *bytecode* (código compilado em linguagem intermédia), bem como outros recursos da aplicação tais como sons e imagens armazenados com um algoritmo de compressão baseado no algoritmo ZIP. O outro ficheiro, com extensão JAD (*Java Application Descriptor*), contém informação acerca do ficheiro JAR associado, nomeadamente, informação acerca do fabricante e da versão da aplicação, dimensão do arquivo JAR associado, entre outras. Quando a aplicação é assinada, a marca resultante do processo de assinatura digital é adicionada ao ficheiro JAD, conforme ilustrado no Exemplo 14:

Exemplo 14 – Conteúdo de um ficheiro JAD, correspondente a uma aplicação com assinatura digital

```
MIDlet-Name: NFC Content Suite
MIDlet-Version: 1.0.0
MIDlet-Vendor: Movensis
MIDlet-Jar-URL: NFCContentSuite-Nokia6131_NFC-pt_PT.jar
MIDlet-Jar-Size: 285132
MIDlet-Description: NFC Content Suite - Content Browser
MIDlet-Icon: /logo.png
MIDlet-Info-URL: http://www.movensis.com/ContentSuite
MIDlet-1: MainMidlet,,com.movensis.midlets.NFCContentSuiteMidlet
MIDlet-Delete-Confirm: Deseja realmente apagar a aplicação?
MIDlet-Permissions:
javax.microedition.io.Connector.http,javax.microedition.io.Connector.bluetooth.client(...)
MIDlet-Push-1:
ndef:external_rtd?name=urn:nfc:ext:movensis:contentsuitedata,com.movensis.midlets.NFCContentSu
iteMidlet,*
MIDlet-Jar-RSA-SHA1: K1lZYxJH7Wu3HKkyxIlpGfym/(...)
MIDlet-Certificate-1-1: MIIFHDCCBASgAwIBAgIQJV(...)
MIDlet-Certificate-1-2: MIIEvzCCBCigAwIBAgIQQZ(...)
```

Neste exemplo é identificável, para além da marca (identificada pelo nome "Midlet-Jar-RSA-SHA1"), dois outros itens relacionados com a assinatura: "Midlet-Certificate-1-1" e "Midlet-Certificate-1-2". Dependendo do certificado utilizado para efectuar a assinatura digital, pode existir apenas um destes itens. Nesse caso, esse item contém o certificado codificado em base64, com o qual a assinatura foi efectuada. A razão da existência de um segundo item neste caso, prende-se com o facto de a CA à qual foi adquirido o certificado utilizar uma CA intermédia para a emissão deste tipo de certificados. Desta forma, para obtermos uma cadeia de certificação válida, é necessário obter o certificado da CA intermédia, que se encontra codificado no segundo item.

A assinatura digital é efectuada sobre todo o conteúdo do ficheiro JAR e o resultado da assinatura (a marca) é adicionado ao ficheiro JAD. Quando a aplicação é instalada, o AMS verifica a assinatura do ficheiro e exibe o nome da entidade que produziu. Desta forma, garante-se que o conteúdo do arquivo não foi adulterado desde a sua assinatura, dando a garantia ao utilizador que o ficheiro que está a instalar foi de facto produzido pela entidade identificada. Garante também que esse ficheiro está em conformidade com o que foi estipulado pelo fabricante do *software*.

5.6.5 Distribuição e instalação da aplicação

Uma preocupação que foi tida em conta para a aplicação em questão foi a forma como esta seria distribuída, quando o sistema entrasse em funcionamento. Tal como foi referido anteriormente, existem várias formas de instalação de aplicações em dispositivos com suporte para a plataforma *Java*. São exemplos de formas de instalação:

- Através de tecnologia de conectividade local disponível no dispositivo.
    - *Bluetooth*, cabo de dados ou Infra-Vermelhos.
- Utilizando uma tecnologia de conectividade à Internet, com instalação suportada por um sistema de distribuição OTA.
    - GPRS, *Wireless Ethernet ou Bluetooth* (se o dispositivo suportar o perfil PAN[1])

[1] Este perfil permite que um dispositivo *Bluetooth* obtenha conectividade à Internet através de um ponto de acesso *Bluetooth*.

Os cenários de conectividade local seriam adequados apenas para os utilizadores que se encontram familiarizados com a instalação de aplicações por estes meios, uma vez que, para além de existir a necessidade de proceder ao *download* prévio da aplicação, pode implicar a instalação de *software* específico do fabricante do dispositivo móvel, bem como a manipulação do mesmo, o que pode ser complicado para certos grupos de utilizadores. Estes cenários seriam bastante úteis para o caso em que a instalação é efectuada num local de exibição de conteúdos, com apoio de pessoas treinadas/formadas para o efeito. No entanto, grande parte dos utilizadores prefere a instalação automática através da Internet, uma vez que o processo é mais simples e tipicamente mais rápido. Para que este tipo de instalações seja possível é necessário configurar um servidor de distribuição OTA, que será apresentado de seguida.

5.6.5.1 Distribuição e instalação baseada num sistema OTA

A tecnologia de distribuição OTA permite que uma aplicação *Java* ME seja instalada directamente através do *browser* do dispositivo móvel, bastando para isto que os ficheiros se encontrem num servidor HTTP adequadamente configurado. Para este modelo de instalação funcionar, quando são efectuados pedidos HTTP GET para os ficheiros (JAR e JAD) relacionados com a aplicação, a resposta deve vir marcada com o tipo MIME correcto. Os tipos MIME que devem ser associados a cada um destes ficheiros estão identificados na Tabela 2.

Extensão	*Tipo MIME*
JAD	text/vnd.sun.j2me.app-descriptor
JAR	application/java-archive

Tabela 2 – Extensões e respectivos tipos MIME necessários num servidor de distribuição OTA

Para que o fluxo descrito na Figura 49 seja possível, é ainda necessário que o atributo Midlet-Jar-URL do ficheiro JAD possua o *Uniform Resource Locator* (URL) absoluto para o ficheiro JAR:

Exemplo 15 – Excerto de um ficheiro JAD, onde é identificada a localização do ficheiro JAR associado

```
MIDlet-Name: NFC Content Suite
(...)
MIDlet-Jar-URL: http://www.movensis.com/ota/NFCContentSuite-Nokia6131_NFC-pt_PT.jar
(...)
```

Depois de configurado o servidor, o processo desenrola-se da forma ilustrada na Figura 49:

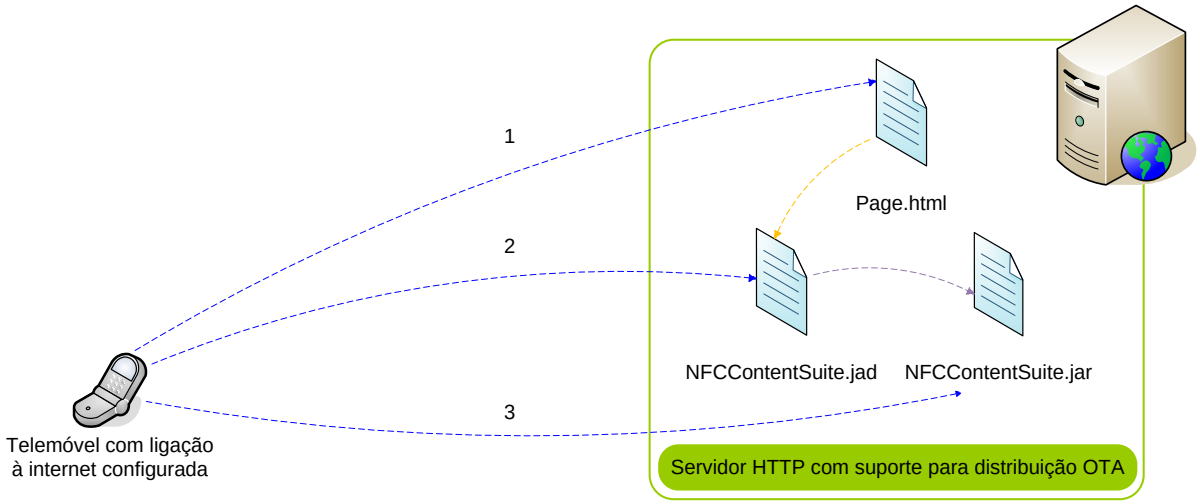


Figura 49 – Processo de instalação típico utilizando um sistema de distribuição OTA

De seguida são apresentados os detalhes de cada um dos passos identificados na Figura 49:

1. O utilizador navega para a página de *download* da aplicação, utilizando o *browser* do dispositivo.
2. De seguida é seleccionada a hiper-ligação de instalação, sendo efectuado o *download* do ficheiro JAD que contém informação acerca do ficheiro JAR associado. Aqui são importantes os atributos que indicam a dimensão, nome, fabricante e localização absoluta da aplicação. Neste passo, é exibida tipicamente uma mensagem onde é apresentada a dimensão do ficheiro JAR, para que o utilizador possa decidir se quer ou não efectuar o *download* da aplicação.
3. Se o utilizador decidir instalar a aplicação, o AMS encarrega-se de efectuar o *download* do ficheiro JAR e proceder à respectiva instalação.

5.6.5.2 Instalação da aplicação de forma automática, utilizando mensagens NDEF

Um dos tipos NDEF RTD suportados por todos os dispositivos NFC é o *Smart Poster* (urn:nfc:wkt:Sp). Este tipo de *record* define o formato binário de mensagens que contêm um *record* do tipo URI (urn:nfc:wkt:U), uma descrição na forma de *record* do tipo Text (urn:nfc:wkt:T), e *record* de um tipo local ao Smart Poster RTD com a identificação da acção a realizar quando o registo for lido (*Recommended Action Record*)[10].

Desta forma, se o URI escrito na *tag* indicar a localização do ficheiro JAD da aplicação, podem ser preparadas *tags* cuja função é a de provocar a instalação da *midlet*, utilizando o sistema de distribuição OTA descrito anteriormente.

Apesar de a utilização descrita anteriormente ser aplicável por si só e servir para resolver o problema, continua a existir a necessidade da escrita de uma *tag* que não possui conteúdo útil da campanha ou exposição em questão. Em cenários como o de um museu, onde existem muitas *tags* com conteúdo na mesma zona isto não é problema porque bastam algumas destas *tags* gravadas de forma especial para suportar a instalação da aplicação em todos os clientes. No entanto, em cenários de elevada dispersão geográfica das *tags*, esta forma de instalação deixaria de ser tão interessante porque, no limite, seria necessário o dobro das *tags* (a que armazena o conteúdo e a que provoca a instalação da aplicação).

Tirando partido do facto que cada mensagem NDEF pode conter vários registos, podemos criar mensagens que contenham para além do *well known record type* NDEF *Smart Poster*, o *record* que contém os dados seriados e que é utilizado pela aplicação desenvolvida (*External* RTD, do tipo urn:nfc:ext:movensis:contentsuitedata).

Quando a *tag* é detectada, o dispositivo NFC começa por tentar interpretar o conteúdo do primeiro registo. Se este for de um dos tipos RTD suportados directamente pelo *firmware*, é realizada a acção associada. Caso contrário, é verificado se o AMS possui alguma aplicação registada para o tipo RTD em questão. Neste caso, o AMS inicia a aplicação registada para esse tipo, e envia-lhe o conteúdo da mensagem NDEF.

Caso o AMS não possua uma aplicação registada para o tipo RTD indicado, é analisado o *record* que se segue na mensagem e repetido todo o processo.

Caso nenhum dos *records* seja interpretado (pelo *firmware* ou por uma aplicação) então o dispositivo pode ignorar a mensagem, ou exibir uma mensagem de informação ao utilizador.

Tendo em conta esta sequência de acções, podemos escrever uma mensagem NDEF que provoque a instalação da aplicação (utilizando o tipo RTD *Smart Poster*) caso o dispositivo ainda não possua a aplicação instalada, e que provoque o arranque da aplicação caso esta já se encontre instalada. Para tal, basta que os *records* estejam ordenados da seguinte forma:

- *Record* que possui os dados da aplicação (*External* RTD)
    - Este *record* é ignorado por dispositivos que não conheçam este tipo e é utilizado para provocar o arranque automático da aplicação quando esta já se encontra instalada.
- *Record Smart Poster* que contém o *record* URI para a instalação da aplicação
    - Este *record* é ignorado por dispositivos que sabem interpretar o primeiro *record* e exibido pelos restantes, provocando a exibição do URI.

Esta organização encontra-se representada na Figura 50.

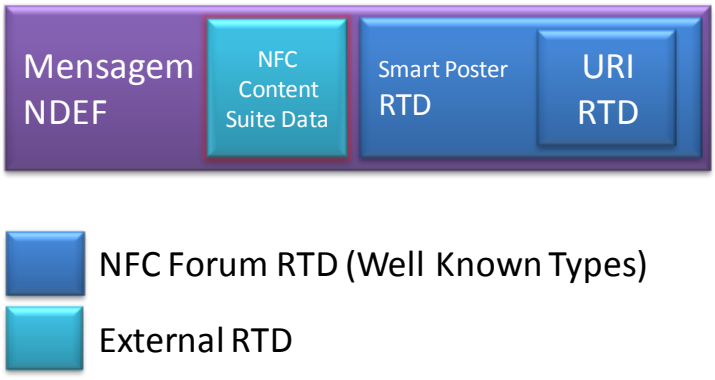


Figura 50 – Organização das mensagens NDEF que suportam a instalação automática da aplicação

5.6.6 Criação de *Streams* para encapsular a escrita e leitura de dados através de NFC-IP1

Um dos requisitos da aplicação é o envio e recepção de informação através da utilização do protocolo NFC-IP1. O suporte para o protocolo NFC-IP1 é um extra que não se encontra coberto pela especificação JSR-

257[25]. No entanto o protótipo utilizado para a implementação deste projecto inclui extensões a esta API de forma a ser suportado.

A API disponível no dispositivo permite apenas o envio de uma mensagem de dimensão até 252 bytes, o que se revela insuficiente perante o cenário do envio de conteúdos de dimensão elevada.

De forma a desacoplar o motor da aplicação deste detalhe, e de acordo com a arquitectura descrita anteriormente, foram modelados dois *streams*:

1. `NFCIP1InputStream`
2. `NFCIP1OutputStream`

Ambos os *streams* são responsáveis pelo envio e recepção da informação em pequenos blocos com dimensão máxima permitida e pela gestão de uma ligação do tipo `com.nokia.nfc.p2p.NFCIPConnection` existente no dispositivo. O primeiro tipo de *stream* é configurado em modo *Target* e o segundo em modo *Initiator*. Um dispositivo *Initiator* é responsável por iniciar a troca de informação e é o primeiro a enviar dados. O *Target* só envia dados em resposta aos envios do *Initiator*.

É também criada uma abstracção em termos das operações necessárias ao envio da informação, nomeadamente:

1. Por cada operação de escrita, tem de existir uma de leitura – este requisito resulta do próprio protocolo NFC-IP1[32][6].
2. Cada vez que os dispositivos ficam fora de alcance um do outro, é lançada uma excepção e a partir desse momento, a ligação actual deixa de ser utilizável sendo necessário criar uma nova.
3. Quando está a ser processada muita informação, por vezes nem todo o buffer que é passado ao método `write` é recebido com sucesso pelo outro terminal. Isto foi verificado durante a implementação deste projecto, e não foi possível apurar as causas deste comportamento.

Os dois últimos problemas identificados são resolvidos tirando partido do primeiro. O facto de ser necessário efectuar uma escrita por cada leitura no caso do *stream* de entrada e vice-versa, é aproveitado para envio de informação acerca da última transferência efectuada. Assim, quando é recebida informação, é enviada a informação de quantos *bytes* foram recebidos. No outro terminal, esta informação é recebida após a escrita e comparada com a informação que foi enviada. Assim, é possível validar se é necessário reenviar parte ou a totalidade do *buffer* anterior.

A modelação desta funcionalidade na forma de *streams* permite a utilização do protocolo NFC-IP1 como fonte de dados adicional para o motor de deseriação, sem qualquer alteração neste último. Outra funcionalidade que é suportada com recurso a estes *streams* é o envio de informação armazenada para outro utilizador. Neste caso, basta ler o ficheiro e escrever o seu conteúdo para o *stream* NFC-IP1 de saída.

5.6.7 Ambiente de desenvolvimento

O ambiente de desenvolvimento utilizado é baseado na plataforma de desenvolvimento *Eclipse*. De forma a facilitar o desenvolvimento de aplicações *Java ME*, é utilizado o produto *J2Me Polish*, que é composto por um conjunto de ferramentas desenhadas para facilitar o desenvolvimento de aplicações para dispositivos móveis[34].

5.6.7.1 *J2Me Polish*

A plataforma *Java ME* é uma plataforma de funcionalidades limitadas quando comparada com outros ambientes de desenvolvimento existentes. Por exemplo, apesar de ser possível obter informação de tipo em tempo de execução, não existe uma *Framework* de seriação como noutras plataformas de desenvolvimento. Esta limitação é razoável, tendo em conta todo o tipo de dispositivos que suportam esta plataforma.

Este conjunto de ferramentas suporta a maioria das funcionalidades extra através de pré-processamento do código produzido pelo programador que as utiliza. Após este pré-processamento, o código é compilado utilizando as ferramentas de compilação comuns. Em alguns casos, no entanto, são utilizados passos de pós-processamento do código em linguagem intermédia gerado pelo compilador.

Entre as diversas ferramentas que compõem este produto, encontram-se ferramentas que tornam possíveis as funcionalidades que se apresentam de seguida. Serão apresentadas todas as ferramentas que foram utilizadas para a implementação deste projecto. Quando possível são apresentados exemplos de utilização neste projecto das ferramentas em questão.

5.6.7.1.1 Ambiente de compilação automática

O ambiente de compilação automático é suportado pela ferramenta *open-source* Ant[35] e configurável através de *scripts* descritos em formato XML.

Esta ferramenta permite a definição de vários conjuntos de passos de compilação (*targets*). Cada *target* pode conter chamadas a ferramentas externas, definição de variáveis, exibição de mensagens, manipulação de ficheiros, ou outras operações. O Exemplo 16 apresenta a definição de um *target*.

Exemplo 16 – *Target* Ant

```
<target name="setdebugbuild" description="(...)">
	<property name="debugbuild" value="true" />
</target>
```

Este *target* pode ser chamado antes do *target* principal de compilação, sendo utilizado para definir um símbolo que indica se o processo de compilação actual deve resultar numa versão de *debug* da aplicação. Este símbolo pode ser utilizado para efectuar compilação condicional, como será descrito no próximo tópico.

A extensibilidade da ferramenta é baseada no conceito de *task*. A *task* é definida através de uma ou mais classes *Java* e é tipicamente configurável. O ambiente de compilação da ferramenta *J2Me Polish* é suportado por uma *task* que contém vários parâmetros configuráveis. O Exemplo 17 apresenta a forma como esta *task* é importada para o processo de compilação actual.

Exemplo 17 – Importação da *task* de compilação J2Me Polish

```
<taskdef name="j2mepolish"
	classname="de.enough.polish.ant.PolishTask"
	classpath="${polish.home}/lib/enough-j2mepolish-
build.jar:${polish.home}/lib/jdom.jar"/>
```

Após a importação da *task*, para o *script* actual, a mesma pode passar a ser utilizada como as restantes *tasks* existentes na ferramenta Ant.

5.6.7.1.2 Compilação condicional

Esta funcionalidade é baseada em propriedades e suporta expressões compostas (condições utilizando várias propriedades, por exemplo).

O modo de funcionamento é muito semelhante ao das directivas de pré-processamento na linguagem C, com a particularidade de ser necessário introduzir a directiva dentro de comentários no código fonte. Isto é necessário para que os ambientes de edição (como o *Eclipse*) que não conhecem este conceito não assinalem erros nestas expressões.

O Exemplo 18 ilustra a utilização desta funcionalidade em conjunto com uma variável definida no *target* que foi mostrado no tópico anterior.

Exemplo 18 – Compilação condicional

```
(...)
//#if debugbuild:defined
BrowserForm.getInstance().append("Processing record...");
//#endif
(...)
```

Apesar de poderem ser definidos vários símbolos para condicionar o processo de compilação, a utilização mais comum desta funcionalidade é em conjunto com a funcionalidade que se descreve no tópico seguinte.

5.6.7.1.3 Base de dados com informação acerca de dispositivos móveis

O ambiente de compilação descrito anteriormente faz uso de uma base de dados que inclui características acerca de vários dispositivos tais como *bugs* conhecidos, especificações técnicas, APIs suportadas, e outros detalhes. Estes dados podem ser utilizados no processo de compilação em conjunto com a funcionalidade de compilação condicional. Isto é conseguido através da definição de variáveis que são mantidas durante o processo de compilação. A definição destas variáveis depende do dispositivo que está actualmente definido como sendo o terminal-alvo da aplicação a ser compilada. Estas variáveis podem depois ser utilizadas durante o processo de compilação condicional, tal como apresentado no Exemplo 19.

Exemplo 19 – Utilização de variáveis obtidas a partir da base de dados de dispositivos

```
(...)
//#if polish.api.nfc
try {
        DiscoveryManager.getInstance().addNDEFRecordListener(
                this,
                new NDEFRecordType(NDEFRecordType.EXTERNAL_RTD,
        "urn:nfc:ext:movensis:contentsuitedata"));
} catch (Exception ex) {
        //#debug
        ex.printStackTrace();
        showErrorMessage(ex,this.menuScreen);
}
//#endif
(...)
```

Neste exemplo, o código que se encontra dentro do bloco //#if só será incluído no ficheiro final que será compilado, se o terminal-alvo do processo de compilação actual suportar a API JSR 257(NFC). Esta base de

dados é composta por ficheiros XML e é extensível. De facto, o símbolo aqui utilizado (suporte NFC) foi adicionado à base de dados de dispositivos no decorrer deste projecto.

5.6.7.1.4 Seriação automática

A *framework* de seriação de dados aqui utilizada baseia-se em pré-processamento e funciona da seguinte forma:

O programador marca o tipo que deseja seriar com a interface `de.enough.polish.io.Serializable`. Durante a fase de pré-processamento, os tipos que implementam esta interface recebem um construtor por omissão (se ainda não existir) e dois métodos:

1. `public void write(DataOutputStream out)`
    a. Este método armazena o conteúdo do objecto no *stream* "`out`"
2. `public void read(DataInputStream in)`
    a. Este método é utilizado para restaurar o estado a partir dos dados presentes no *stream* "`in`"

No corpo destes métodos são colocadas chamadas ao método `serialize` ou `deserialize` do tipo `de.enough.polish.io.Serializer`, dependendo da operação a realizar (`write` ou `read`, respectivamente). São efectuadas tantas chamadas quantos os campos seriáveis que sejam encontrados. O tipo `de.enough.polish.io.Serializer` utiliza um formato próprio para armazenar os tipos primitivos da linguagem. Para campos de tipos compostos, é verificado se o tipo em questão implementa a interface `de.enough.polish.io.Serializable`. Em caso afirmativo, a seriação é realizada através de chamadas aos métodos read e write descritos anteriormente.

Esta *framework* é depois utilizada por outras classes, como é o caso do tipo `de.enough.polish.io.RmsStorage`. Esta classe permite a escrita e leitura de informação do *record store* da aplicação, de forma simplificada, como se ilustra no Exemplo 20. O *record store* é um local onde as aplicações *Java ME* podem armazenar estado de forma persistente. Este é composto por registos, formados por uma sequência de *bytes* e identificados por um valor inteiro.

Exemplo 20 – Utilização da *framework* de seriação através do tipo *RmsStorage*

```
(...)
settingsStorage = new RmsStorage(SETTINGS_RECORDSTORENAME);
for (String s : settingsStorage.list()) {
  if (s==null || "".equals(s))
    continue;
    if (SETTINGS_FILESYSTEM.equals(s)){
      this.fileSystemSettings = (FileSystemSettings)settingsStorage.read(s);
    }
}
(...)
```

5.6.7.1.5 Automatização de tarefas de rotina

Desde que uma aplicação *Java ME* é compilada até à sua distribuição, podem ser necessários vários passos adicionais. Estes passos são tipicamente automatizáveis e este ambiente de compilação disponibiliza formas para automatizar grande parte destas tarefas. Uma delas é o processo de assinatura que é configurado da seguinte forma através de um elemento da *task* Ant descrita anteriormente, como se pode ver no Exemplo 21.

Exemplo 21 – Assinatura da *midlet* utilizando *J2Me Polish*

```
<sign keystore="midlets.jks"
       key="MovipagaSign"
       password="<password>"
       if="sign"
/>
```

Previamente, tem de ser preparado o *keystore* (ficheiro "midlets.jks") que contém a chave privada e o certificado associado, que serão utilizados para gerar e verificar a assinatura da aplicação, respectivamente.

Outro dos passos que é efectuado é a ofuscação da aplicação. O processo de ofuscação serve dois propósitos distintos:

1. Redução da dimensão do código gerado.
2. Aumento da dificuldade na interpretação do código intermédio gerado.

O processo de ofuscação altera os nomes dos tipos, variáveis locais, e *packages*, mantendo no entanto a validade do código. Alguns ofuscadores conseguem ainda verificar que certos membros e tipos não são necessários e eliminam-nos.

O primeiro propósito é obtido porque, (para além da remoção de código desnecessário) os nomes atribuídos tipicamente pelo programador são longos e descritivos. Ao mudar estes nomes para cadeias de caracteres muito menores (na ordem dos 2 caracteres), consegue-se produzir um ficheiro de linguagem intermédia mais pequeno.

O segundo propósito elimina a descrição da semântica utilizada pelo programador, dificultando a interpretação do código por parte de terceiros. Este efeito é desejável, porque o código gerado encontra-se numa linguagem intermédia muito rica que permite obter o código original através da utilização de ferramentas existentes para esse efeito.

O Exemplo 22 apresenta a forma como este passo é configurado.

Exemplo 22 – Configuração do processo de ofuscação

```
<obfuscator name="ProGuard" unless="test || polish.blackberry" >
       <parameter name="include" value="myconfig.pro" />
       <parameter name="optimize" value="true" />
</obfuscator>
```

Um passo menos comum é a criação de ficheiros JAR utilizando ferramentas com performance superior à do compressor utilizado por omissão ("*Java Archive Tool*"). Este passo adicional permite obter ganhos na ordem dos 5KB (no caso da aplicação implementada). Isto é importante, porque quanto maior for o arquivo final, mais memória será necessária para o descomprimir, e mais tráfego será gerado durante a transferência da aplicação.

Outro factor muito importante e que faz com que este tipo de questões seja encarado com muito interesse, é o facto de certos dispositivos possuírem uma dimensão máxima para o ficheiro JAR: a partir de um determinado valor que varia de dispositivo para dispositivo, a aplicação simplesmente não será instalada pelo AMS. Nestes casos é apresentada uma mensagem de erro ao utilizador, e a instalação não pode prosseguir.

Este passo é conseguido através da utilização do elemento de configuração apresentado no Exemplo 23.

Exemplo 23 – Configuração de uma ferramenta de compressão alternativa

```
<packager name="kzip"/>
```

5.6.7.1.6 Motor gráfico com suporte para definição de estilos

Outra das funcionalidades incluídas neste conjunto de ferramentas, é um motor gráfico que adiciona a possibilidade de definir o aspecto da aplicação apenas através de um ficheiro de estilo, utilizando conceitos semelhantes aos das *Cascading Style Sheets* (CSS).

O motor gráfico é parametrizado com propriedades obtidas a partir da base de dados de dispositivos, e através de propriedades definidas num ficheiro denominado "styles.css" que contém um conjunto de classes. Essas classes podem ser pré-definidas ou criadas pelo programador. Quando um controlo é associado a uma classe, é efectuada a definição de propriedades adicionais sobre esse controlo.

Esta secção será demonstrada utilizando um controlo criado durante o desenvolvimento da aplicação de exibição de conteúdos, denominado `FileSystemStatus`, cujo aspecto se apresenta na Figura 51.

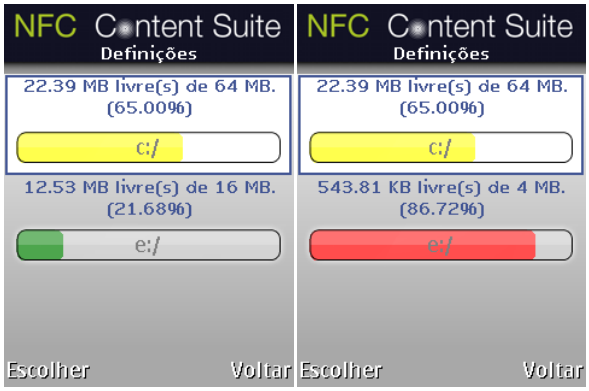


Figura 51 – Aspecto do controlo *FileSystemStatus*

Cada unidade de armazenamento é representada por uma instância deste controlo. Este controlo foi criado para facilitar a exibição do estado das unidades de armazenamento disponíveis no dispositivo. Foi implementado com recurso ao modelo de parametrização da interface gráfica descrito anteriormente. Este controlo apresenta o espaço livre em valor absoluto e em percentagem. Em ambas as imagens da Figura 51, existem duas instâncias deste controlo.

A associação da classe aos controlos é efectuada mais uma vez através de pré-processamento, tal como demonstrado no Exemplo 24.

Exemplo 24 – Atribuição de uma classe a um determinado controlo gráfico

```
(...)
//#style fileSystemStatus
FileSystemStatus fss = new FileSystemStatus(fc.usedSize(), fc.totalSize(),root);
append(fss);
(...)
```

Um controlo que suporte esta funcionalidade deve possuir um construtor adicional que para além dos parâmetros de construção normais, receba uma instância do tipo `de.enough.polish.ui.Style`. Esta instância contém a informação de estilo associada ao controlo em questão, nomeadamente, as propriedades que se encontram definidas e os respectivos valores. Isto é necessário, porque após o pré-processamento da marca `#style`, é adicionado um novo argumento de construção com a informação descrita anteriormente.

Após a definição do nome da classe, deve ser criado um estilo no ficheiro "styles.css" tal como se apresenta no Exemplo 25.

Exemplo 25 – Excerto do ficheiro styles.css com a definição do estilo do controlo *FileSystemStatus*

```
.fileSystemStatus{ /* Identificador da classe */
	(...)
	fsstatus-background-color{
		high:red; /*Quando o nível de ocupação é elevado*/
		medium:yellow; /*Quando o nível de ocupação é médio*/
		low:green; /*Quando o nível de ocupação é baixo*/
	}
	fsstatus-overlay-image:url(fsstatusimage.png); /* Imagem de overlay */
	fsstatus-foreground-color:gray; /* Cor do texto */
	fsstatus-threshold{
		lowtomedium:25; /* % a partir da qual a ocupação é média */
		mediumtohigh:75; /* % a partir da qual a ocupação é alta */
	}
	fsstatus-fill-arc{ /* Características do rectangulo de fundo */
		width:15;
		height:15;
	}
	(...)
}
```

5.6.7.1.7 Ferramentas de localização da aplicação

A localização da aplicação também é suportada pela utilização de pré-processamento, que faz uso de propriedades definidas num ficheiro com o nome "messages.txt", carregado de forma dependente da cultura para a qual se pretende compilar a aplicação. Este ficheiro contém um conjunto de propriedades e o respectivo valor. A Figura 52 apresenta a estrutura de uma aplicação em que existem duas versões do ficheiro "messages.txt". Uma para a linguagem "pt" (português) e outra para a linguagem "en" (inglês). Como neste caso não é especificada uma cultura, qualquer uma que utilize estas linguagens pode ser suportada.

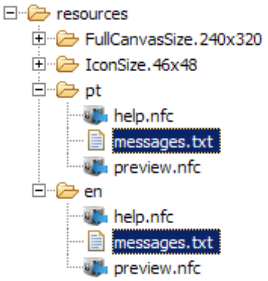


Figura 52 – Organização dos recursos de acordo com a cultura

A especificação de culturas pode ser feita também nesta estrutura de ficheiros. Por exemplo, para o caso de português Europeu, deveria ser criada uma pasta com o nome "pt-pt" e para o caso de português do Brasil, teria de ser criada uma pasta com o nome "pt-br". A estrutura apresentada na Figura 52 suporta a compilação para ambas as linguagens mas, neste caso, ambas as versões finais serão iguais porque não existem recursos específicos para qualquer uma das culturas.

O controlo `FileSystemStatus` apresentado anteriormente utiliza esta funcionalidade para tornar a sua apresentação independente da linguagem. No Exemplo 26 são identificadas as propriedades que são utilizadas por este controlo.

Exemplo 26 – Excerto do ficheiro "messages.txt" com propriedades relacionadas com o controlo `FileSystemStatus`

```
(...)
contentsuite.filestatus.formatstring={0} livre(s) de {1}. ({2}%)
contentsuite.filestatus.units.gb= GB
contentsuite.filestatus.units.mb= MB
contentsuite.filestatus.units.kb= KB
contentsuite.filestatus.units.b= B
(...)
```

A utilização destas propriedades é feita utilizando a classe auxiliar `de.enough.polish.util.Locale`. Para os casos em que se trata de substituições simples, como no caso das últimas 4 propriedades que são apresentadas no Exemplo 26, é feita uma substituição da chamada ao método de tipo `get` da classe `de.enough.polish.util.Locale`, pela string literal correspondente que é obtida no ficheiro "messages.txt" actual. Esta substituição é realizada quando é efectuado o passo de pré-processamento relacionado com a localização. No caso em que a substituição não pode ser directa, como é o caso da primeira propriedade em que é necessária uma substituição através do uso de argumentos, a substituição é efectuada em tempo de execução. A utilização das duas variantes desta funcionalidade encontra-se resumida no Exemplo 27 e no Exemplo 28.

Exemplo 27 – Exemplo de utilização da classe `de.enough.util.polish.Locale` para substituições simples

```
(...)
if (amount >= B_TO_KB_THRESHOLD){
  unit = Locale.get("contentsuite.filestatus.units.kb");
  divisor = B_TO_KB_THRESHOLD;
}
(...)
```

Exemplo 28 – Exemplo de utilização da classe `de.enough.util.polish.Locale` para substituições baseadas em argumentos

```
private String getLabelString(){
  //contentsuite.filestatus.formatstring={0} livres de {1}. ({2}%)
  (...)
  String[] parameters = new String[]{ availStr, totalStr, percentStr};
  return Locale.get( "contentsuite.filestatus.formatstring" , parameters );
}
```

5.6.7.1.8 Escolha condicional de ficheiros de suporte

Os recursos utilizados na aplicação tais como imagens, sons ou vídeos que podem ser incluídos no ficheiro JAR podem necessitar de variar consoante as características do dispositivo: quer pela dimensão do visor, quer pelo suporte para funcionalidades multimédia. O processo de compilação pode incluir estes mesmos recursos de forma condicional, de acordo com as características do dispositivo actual.

Aqui também é possível ganhar em termos de dimensão dos ficheiros finais, porque os ficheiros JAR dos dispositivos que possuam ecrãs mais pequenos são mais pequenos devido à redução das imagens que são incluídas.

A escolha é feita da seguinte forma: se o dispositivo actual possuir a propriedade `FullCanvasSize` com os valores 240x320, então os ficheiros da pasta "FullCanvasSize.240x320" são incluídos. Este modelo permite ainda variantes tais como utilizar apenas parte da propriedade (altura / largura) ou utilizar operadores tais como "+" e "-" que indicam se um determinado conjunto de ficheiros deve ser incluído no caso em que uma propriedade for superior ou inferior ao valor especificado.

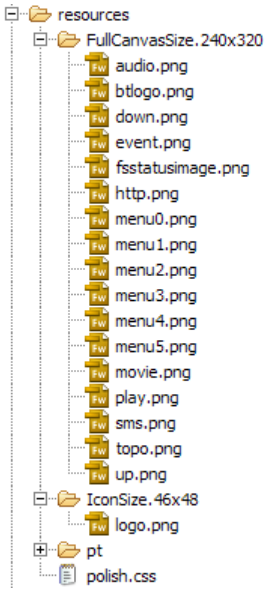


Figura 53 – Estrutura de ficheiros de suporte à escolha condicional

Este modelo permite ainda a utilização da informação de cultura actual para a escolha de recursos.

5.6.7.1.9 Compilação simultânea para vários dispositivos

Existem duas abordagens para a resolução de problemas derivados das especificidades dos terminais móveis nos quais se pretende que a aplicação funcione:

1. Detecção em tempo de execução das características do dispositivo, nos casos em que isto é possível, utilizando apenas um par de ficheiros JAR+JAD
2. Criação de um par de ficheiros JAR+JAD para cada dispositivo ou famílias de dispositivos

A primeira opção é trabalhosa e problemática, porque torna tarefas como o acesso a uma funcionalidade que não esteja contemplada em todos os dispositivos numa tarefa muito complexa.

Antes de surgir o conceito de compilação condicional baseado numa base de dados de dispositivos, a compilação da aplicação para vários dispositivos/famílias de dispositivos levantava outros problemas, nomeadamente a manutenção do código das várias versões e a compilação em simultâneo das mesmas. Isto é suportado pela *Framework J2ME Polish* através da utilização de compilação condicional em conjunto com a base de dados de dispositivos e ficheiro de configuração do processo de compilação.

Neste último, é possível especificar quais os dispositivos-alvo da compilação actual, tal como apresentado no Exemplo 29.

Exemplo 29 – Definição do conjunto de dispositivos para os quais a aplicação deve ser compilada

```
<deviceRequirements>
       <requirement name="Identifier" value="Nokia/6212_NFC" />
       <requirement name="Identifier" value="Nokia/6131_NFC" />
</deviceRequirements>
```

5.6.7.1.10 Funcionalidades Java5

Uma das ferramentas incluídas neste conjunto é o suporte para a utilização de um subconjunto das funcionalidades da plataforma Java5, como por exemplo, a utilização de contentores genéricos, de tipos enumerados e de ciclos `for-each`.

Exemplo 30 – Utilização de funcionalidades da plataforma *Java5* em *Java* ME

```
Vector<NFCInfoControlBase> currentControls = new Vector<NFCInfoControlBase>();

public void clearControls(){
  for(NFCInfoControlBase ctrl : currentControls){
    if (ctrl instanceof StreamedMediaElementControl){
      StreamedMediaElementControl smectrl = (StreamedMediaElementControl)ctrl;
       smectrl.discardControl();
    }
  }
  currentControls.removeAllElements();
  this.deleteAll();
}
```

Esta funcionalidade é suportada através de um passo de processamento do código em linguagem intermédia gerado.

5.6.8 Modelo de carregamento e apresentação de conteúdos

Todos os elementos estendem uma classe abstracta `NFCInfoElement`, que se encontra representada na Figura 54.

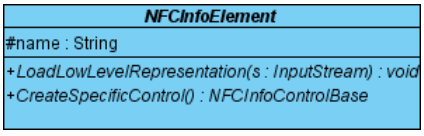


Figura 54 – Classe *NFCInfoElement*

Cada tipo de elemento é responsável por carregar a sua própria representação em memória a partir de um *stream* de entrada. Este passo é realizado na implementação do método `LoadLowLevelRepresentation`. Quando este método é chamado pelo motor de seriação, significa que foi identificada a presença de um elemento de informação do tipo específico em questão. A partir deste momento, o elemento deve obter o número de propriedades que o *stream* de entrada possui, bem como os seus respectivos valores.

Em termos de representação visual, cada tipo de elemento possui um tipo de controlo gráfico associado. Esta associação é realizada pelo próprio elemento. Todos os elementos que podem ser carregados implementam o método `CreateSpecificControl`, no qual retornam uma instância do controlo que representa o elemento em questão de forma gráfica. Todos os controlos gráficos que representam elementos de informação estendem a classe `NFCInfoControlBase` (ver Figura 55). Esta classe estende o tipo `javax.microedition.lcdui.CustomItem`, o que significa que o desenho do controlo é da responsabilidade de cada um dos controlos implementados.

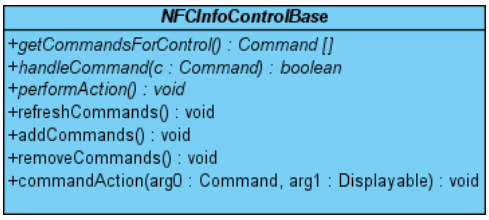


Figura 55 – Classe `NFCInfoControlBase`

Todos os controlos possuem um conjunto de comandos que podem ser executados. Estes são obtidos através da chamada ao método `getCommandsForControl`. Estes comandos ficam disponíveis quando o controlo gráfico associado estiver seleccionado. Adicionalmente, todos os controlos têm uma acção que é realizada por omissão, e que é invocada quando o utilizador selecciona a tecla central de navegação (se disponível). Esta acção é associada ao método `performAction` de cada controlo.

5.6.8.1 Descarregamento e apresentação de conteúdos multimédia

Devido a limitações de capacidade de memória de execução dos dispositivos que suportam a plataforma *Java ME*, não é possível carregar conteúdos de dimensões elevadas (geralmente vídeos ou áudio) da mesma forma que os restantes (imagens, texto ou outros tipos de informação de dimensão reduzida). Para resolver este problema a solução passa pelo envio do conteúdo à medida que vai sendo necessário (*streaming*). Este modo de operação é suportado pelos dispositivos que suportam a MMAPI [15].

Para tornar possível a entrega de conteúdos com estas características, os elementos de informação multimédia (áudio/vídeo) contêm apenas o identificador do conteúdo multimédia associado. Quando o utilizador decide abrir o conteúdo é efectuado o pedido ao servidor de conteúdos associado, este é guardado localmente no dispositivo móvel e finalmente é exibido. O armazenamento local é necessário para um conjunto de dispositivos que apenas suportam a leitura de informação em modo *streaming* se a fonte de dados for um ficheiro no sistema de ficheiros. O dispositivo utilizado para o desenvolvimento deste projecto também tem esta limitação.

5.6.8.2 Utilização do motor de carregamento e exibição de conteúdos para outros fins

A flexibilidade conseguida na implementação do motor de carregamento e exibição de conteúdos permitiu a sua utilização noutros contextos dentro da aplicação. Um exemplo disto é a implementação do menu de ajuda. Neste caso, existe um ficheiro denominado "help.nfc" que é incluído no ficheiro JAR que contém a aplicação. Este ficheiro foi criado utilizando a aplicação de edição de conteúdos e a funcionalidade de gravação do conteúdo para ficheiro. Como tal, está de acordo com o protocolo de seriação descrito anteriormente. Quando o utilizador selecciona este menu, o motor de exibição de conteúdos carrega o conteúdo a partir desse ficheiro, e o utilizador pode depois navegar na ajuda tal como navega num conteúdo normal. Este menu é apresentado na Figura 56.

Figura 56 – Menu de ajuda da aplicação

Para facilitar a criação de conteúdos e a respectiva pré-visualização no dispositivo móvel, foi criada a possibilidade de utilizar a aplicação de visualização de conteúdos para ver a informação que está a ser editada no editor de conteúdos. Isto permite que o gestor de conteúdos do sistema valide imediatamente a forma como os conteúdos serão apresentados aos utilizadores da solução. Foram criadas duas formas de comunicação para esta funcionalidade: NFC-IP1 (apresentado anteriormente) e *Bluetooth*.

A primeira não exigiu alterações na aplicação de visualização de conteúdos.

Para a segunda, foi necessário criar uma versão especial da aplicação, na qual é apresentado um novo menu. A acção associada a este menu é o carregamento do ficheiro "preview.nfc" que, à semelhança do ficheiro utilizado para o menu de ajuda, reside no ficheiro JAR que contém a aplicação. Este ficheiro foi criado com a aplicação de edição de conteúdos, e contém apenas um elemento de ligação *Bluetooth*. Este elemento de ligação está configurado de forma a efectuar a ligação de imediato, utilizando um identificador de serviço especial.

A aplicação de edição de conteúdos por sua vez publica o serviço associado ao identificador de serviço referido anteriormente. Os componentes que dão suporte a esta funcionalidade são apresentados na Figura 57.

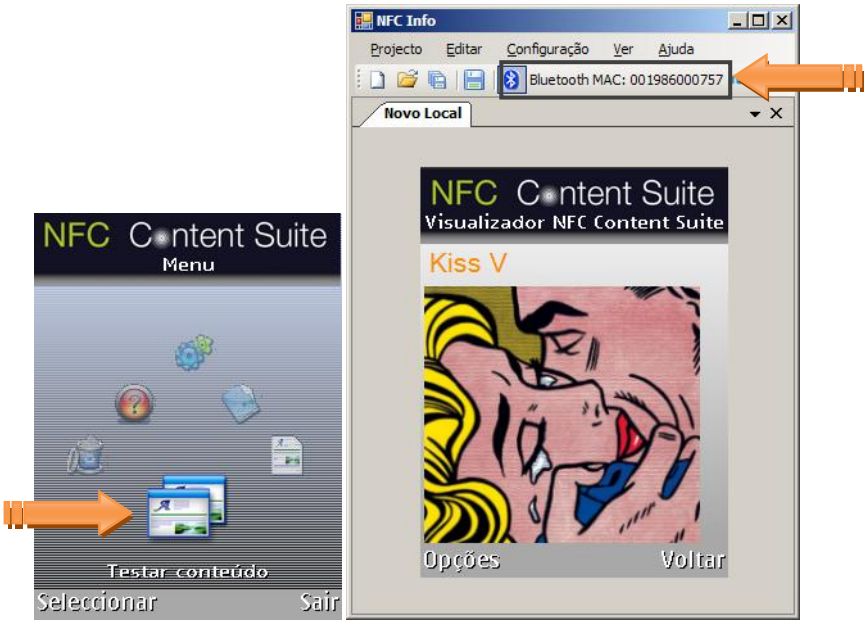


Figura 57 – Componentes que suportam a pré-visualização de conteúdos através de *Bluetooth*

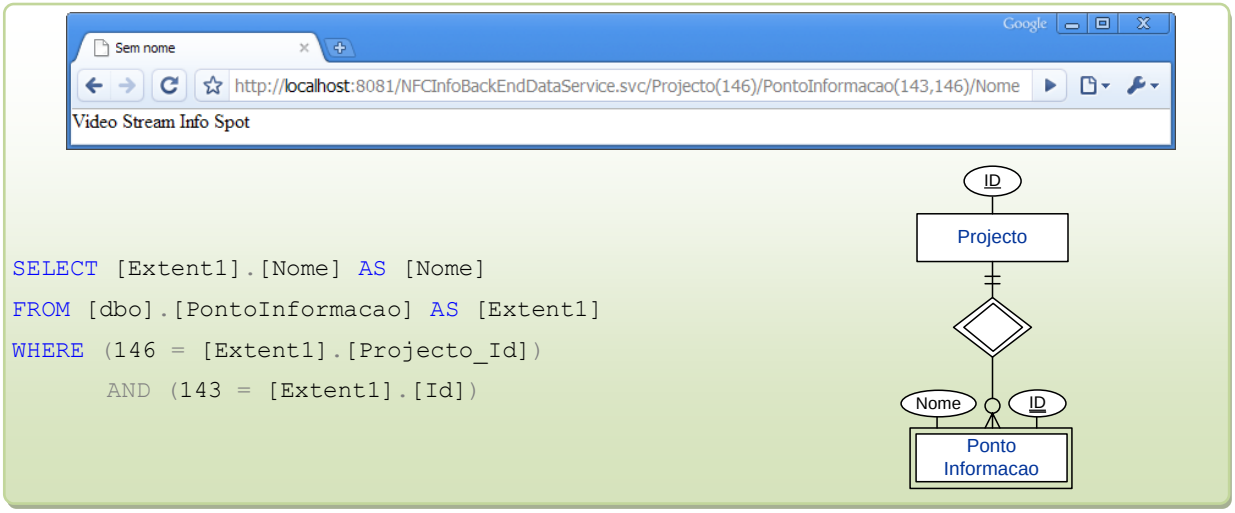
5.7 Web Service para acesso ao servidor de dados

Para dar suporte ao acesso à Base de Dados em configurações com alguns componentes ligados entre si através da Internet, está a ser utilizada uma interface *REpresentational State Transfer* (REST), criada com a tecnologia *ADO.NET Data Services*. Esta interface baseia-se no conceito de que qualquer recurso pode ser identificado de forma única, através de um URL. Neste caso, os recursos são células de uma tabela numa base de dados.

Esta tecnologia gera de forma automática um serviço que traduz os pedidos HTTP efectuados, em acções sobre a base de dados. A escolha da informação é feita de acordo com o endereço que é pedido ao serviço, e a operação pretendida sobre os dados é definida através da utilização dos verbos do protocolo HTTP (GET, PUT, POST e DELETE).

Apesar de este modelo suportar as quatro acções básicas de armazenamento persistente (CRUD – *Create Read Update and Delete*), para este projecto apenas foi necessária a leitura de informação. O Exemplo 31 ilustra a forma como é efectuado o mapeamento entre pedidos REST de GET (que estão associados a operações de leitura) e interrogações à base de dados. Neste caso é obtido o nome do Ponto de Informação (Entidade Fraca de Projecto) com a chave composta "143,146". Este ponto de informação está associado ao Projecto com identificador 146. A interrogação apresentada foi criada de forma automática pelo serviço de acesso a dados, com base no URL apresentado. Os dados são transportados utilizando *Java Script Object Notation* (JSON)[36] e *eXtensible Markup Language* (XML)[37].

Exemplo 31 – Mapeamento entre pedidos REST [GET] e interrogações à base de dados



Esta interface tem ainda a vantagem de se integrar com as novas funcionalidades da linguagem C# 3.0, possibilitando assim a realização de interrogações de uma forma semelhante a SQL, como ilustrado no Exemplo 32.

Exemplo 32 – Utilização de C#3.0 para efectuar interrogações ao serviço de dados

```
var Nome = from pontoInformacao in BackEnd_DataContext.PontoInformacao
           where pontoInformacao.Projecto_Id == 146 && pontoInformacao.Id == 143
           select new { NomePontoInformacao = pontoInformacao.Nome };
```

Uma vez que o serviço fica disponível na Internet, é necessário definir quais as tabelas que estão visíveis através deste serviço e qual o conjunto de acções permitido sobre cada tabela da base de dados. No caso desta

aplicação, foi apenas concedido acesso às tabelas necessárias para a obtenção dos conteúdos por parte dos servidores de conteúdos.

5.8 Servidores de conteúdos

Aqui serão apresentados os detalhes relacionados com a implementação dos servidores de conteúdos, cujos conceitos foram apresentados no Capítulo 3.

5.8.1 Bluetooth

Este servidor encontra-se implementado em modo consola e utiliza uma biblioteca *shared source* denominada *32 Feet .NET* [38], para a comunicação *Bluetooth*. Desta biblioteca foi utilizado o suporte para a publicação de serviços e para o envio e recepção de dados.

Após a publicação do serviço com o identificador especificado (conhecido pela aplicação de visualização de conteúdos), o modo de funcionamento deste servidor resume-se ao seguinte: quando um cliente se liga, envia o identificador do conteúdo. Com este identificador a aplicação acede ao *Web service* de acesso a dados apresentado anteriormente, obtém o conteúdo e envia-o para o cliente.

5.8.2 HTTP

Este servidor foi modelado como um *handler* ASP.NET (*Active Server Pages .NET*)[39] que recebe argumentos via *query string*, acede ao *Web service* de acesso a dados e envia para o cliente o conteúdo, obedecendo ao formato de transferência de dados definido anteriormente. ASP.NET é a plataforma da *Microsoft* para desenvolvimento de aplicações *Web*. Para criar um *handler* ASP.NET é definida uma classe que implementa a interface `System.Web.IHTTPHandler`. Um dos métodos desta interface (`ProcessRequest`) é chamado pela *Framework* cada vez que é feito um pedido HTTP de GET que obedeça a determinadas condições. Estas condições são definidas através da configuração da *Framework* ASP.NET. No caso desta aplicação foi definido que seja qual for o URI do pedido de GET, desde que a extensão do ficheiro pedido seja ".nfc", o pedido é encaminhado para este *handler*. Esta opção foi tomada para reduzir a dimensão do URI que tem de ser guardado nas *tags* para possibilitar a realização do pedido. Com esta configuração, o URI http://servidor/.nfc?id=idconteudo provoca a execução do *handler*.

Para cada pedido, é obtido o identificador a partir da *query string* e de seguida é consultado o *Web service* de acesso a dados para obter o conteúdo associado ao identificador recebido. Este conteúdo é depois enviado para o cliente, que neste caso se encontra a executar a aplicação de visualização de conteúdos.

5.9 Sumário do capítulo

Neste capítulo foram apresentados os principais aspectos de implementação dos componentes da solução desenvolvida.

A parte inicial deste capítulo foi dedicada aos componentes relacionados com a persistência de informação em base de dados. A apresentação do servidor de dados introduziu conceitos relacionados com a forma como é feita a associação dos conteúdos a projectos. Estes conceitos são depois contextualizados com a

organização escolhida para o armazenamento dos dados. Ainda relacionado com a persistência de informação em base de dados, foi apresentada a forma como foi obtida a camada de acesso a dados que facilita a obtenção dos dados da base de dados.

De seguida, foi apresentado o motor de seriação de dados e a forma como é utilizado o suporte para informação de tipo em tempo de execução da plataforma .NET para implementar o modelo de transferência de dados apresentado na Secção 3.4.

Relativamente à aplicação de edição de conteúdos, foi apresentada a sua organização, identificando os componentes mais relevantes e as interacções que ocorrem entre eles. Foi também identificada a forma como os conceitos apresentados no capítulo 3 se relacionam com as funcionalidades disponíveis na aplicação. Esta secção refere as bibliotecas que foram criadas para interagir com o leitor da Arygon que é utilizado nesta solução, e que permitem a escrita de *tags Mifare* e envio de informação utilizando o protocolo NFC-IP1. A apresentação destas bibliotecas é precedida pela apresentação técnica do leitor NFC, necessária ao suporte de algumas decisões tomadas na implementação.

Outro dos componentes apresentado foi a aplicação de visualização de conteúdos. Nesta secção, para além dos detalhes mais relevantes da implementação, foi apresentado o enquadramento da aplicação na solução desenvolvida e apresentada a plataforma de *software* e de *hardware* utilizada para o desenvolvimento da aplicação. Durante a apresentação da plataforma de *software* foram apresentados exemplos da utilização das funcionalidades à medida que estas foram introduzidas.

Finalmente, foi apresentada a forma como se encontram implementados os servidores de conteúdos que dão suporte ao envio de conteúdos de dimensões elevadas. Nesta apresentação é contextualizada a utilização do *Web service* para acesso a dados nos servidores de conteúdos.

Capítulo 6

6 Resultados e Trabalho Futuro

Este capítulo é dedicado à análise dos objectivos propostos e de que forma foram ou não alcançados. A forma escolhida para esta análise é através da comparação entre o sistema desenvolvido e as soluções existentes no mercado. Tendo sido previamente identificados os pontos fortes e fracos de cada uma delas, esta análise é feita com base nessas observações. É também analisada a flexibilidade da solução desenvolvida, através da identificação da configuração necessária no sistema para suportar as funcionalidades identificadas nos projectos relacionados. No fim deste capítulo são identificados os tópicos para trabalho futuro.

6.1 Resultados

Para o caso do projecto de Caen, seria necessário suportar a realização de chamadas e envio de mensagens de SMS de forma automática. Em relação às chamadas de voz, utilizando o sistema desenvolvido, poderia ser criado um documento que contivesse um elemento URI com endereço do tipo tel:[número]. Quando este elemento fosse seleccionado, seria efectuada a chamada. Esta solução tem a vantagem de suportar a apresentação de mais informação no mesmo documento em que é apresentada a opção de efectuar a chamada. Tem a desvantagem de necessitar a instalação prévia da aplicação. No entanto, utilizando o modelo de distribuição apresentado anteriormente, desde que exista ligação à Internet disponível no terminal, a instalação é simples. Outra vantagem desta solução é a possibilidade de guardar o conteúdo para consulta ou partilha posterior.

Em relação ao envio de mensagens de SMS, seria possível uma abordagem semelhante, mas desta vez utilizando um elemento de envio de SMS. Neste elemento seria definido qual o formato da mensagem a enviar, bem como o número de destino e o porto de entrega da mensagem. Neste caso, as vantagens e desvantagens são iguais às anteriores. No entanto, uma vez que as mensagens de SMS são utilizadas na solução de Caen apenas para obter informação na forma de outra mensagem de SMS, esta solução poderia ser substituída pela colocação dessa mesma informação num documento utilizando elementos de texto formatado. Assim, o conteúdo seria apresentado de forma mais adequada.

Para o caso do projecto do percurso Princesa Grace do Mónaco, seria apenas necessária a criação de conteúdo e configuração do servidor de conteúdos HTTP para conseguir funcionalidade semelhante. No entanto, é possível colmatar algumas das falhas identificadas quando foi feita a análise do projecto. Como o percurso tem localização fixa e uma dispersão geográfica bem definida, poderia ser utilizado o servidor de conteúdos *Bluetooth*. Isto resolveria dois problemas: o custo e lentidão associados às comunicações GPRS. Através da utilização de servidores de conteúdos *Bluetooth*¸ seria possível entregar conteúdos multimédia. Esta funcionalidade poderia ser utilizada neste caso para apresentar excertos de documentários e entrevistas ou apresentar um documento com informação textual, mantendo música a tocar durante a leitura do mesmo. Outra das vantagens da solução é a possibilidade de gravar o conteúdo localmente para consulta posterior.

Tendo em conta os dois exemplos utilizados nesta análise, podemos concluir que a solução desenvolvida é suficientemente flexível não só para suportar os cenários de aplicação de outras soluções do mesmo género, mas também para melhorar a usabilidade destas. Outra das vantagens desta solução perante as soluções analisadas prende-se com a capacidade de entrega de conteúdos multimédia.

Os objectivos de suporte para gestão e distribuição de conteúdos de dimensões elevadas utilizando ligações disponíveis no dispositivo móvel, estabelecidos inicialmente, são suportados pelos conceitos de elementos de ligação e servidor de conteúdos. Para além destes cenários mais complexos, a solução alcançada permite a implementação de cenários simples, em que o conteúdo é lido apenas de uma *tag* NFC, ou de cenários mistos em que o utilizador pode consultar um resumo das informações, e com base no conteúdo do resumo, optar por obter mais informações utilizando uma ligação *Bluetooth* ou HTTP.

6.2 Trabalho futuro

Durante a realização deste projecto foram identificadas determinadas funcionalidades que, apesar de não serem essenciais para garantir a aplicabilidade da solução, seriam adequadas ao sistema em questão.

No cenário actual, em que o Universo de dispositivos móveis que suportam NFC é limitado, o editor gráfico de conteúdos assume que a dimensão do ecrã do dispositivo alvo é de dimensão fixa, e que apenas é suportada a entrega de conteúdos para dispositivos com essas características. Um ponto possível para melhoria desta solução seria o suporte de criação de perfis de dispositivos que partilham determinadas características em termos de exibição de informação. Desta forma quando fosse editado um conteúdo, sê-lo-ia de acordo com um perfil seleccionado. O motor de exibição apresentado anteriormente adequa-se a esta funcionalidade. Seriam apenas necessárias alterações ao nível da lógica da própria aplicação, de forma a suportar a criação de perfis. Os perfis poderiam ser armazenados no Serviço de Dados.

Ainda na aplicação de edição de conteúdos, poderia ser facilitada a criação de conteúdos com texto formatado. Na versão actual da aplicação quando está a ser criado texto formatado, é necessário criar um elemento de texto cada vez que se deseja alterar a formatação. Uma solução poderia ser a criação de um editor para texto formatado que suportasse a edição do texto de forma mais conveniente, como é tipicamente suportado nos processadores de texto. Este texto seria depois convertido em tantos elementos de texto quantos os necessários, de forma automática.

Também na aplicação de edição de conteúdos, poderia ser dada indicação da dimensão actual do conteúdo a ser editado. Poderia ainda ser efectuada com base nesta dimensão, a recomendação de quais os tipos de *tag* configurados para o projecto actual que poderiam ser utilizados para o conteúdo em causa. Caso a dimensão do conteúdo ultrapassasse a capacidade de armazenamento de todas as *tags* associadas ao projecto, poderia ser sugerida a promoção do conteúdo local a conteúdo remoto. Poderia ainda ser criado de forma automática um documento com um elemento de ligação para acesso ao conteúdo remoto.

A última melhoria que foi pensada para a aplicação de edição de conteúdos é relativa à escrita dos conteúdos criados na aplicação em *tags* NFC. Actualmente, a escrita de conteúdos é apenas suportada em *tags Mifare*. Esta alteração é dependente da actualização das bibliotecas de acesso ao leitor NFC. No entanto, a estrutura de suporte à extensão destas bibliotecas está criada, e o suporte para os restantes tipos de *tags* NFC passaria apenas pelo estudo dos protocolos associados a cada uma das restantes famílias de *tags*, e chamada dos comandos necessários do micro-controlador PN531 incluído no leitor Arygon.

À semelhança das soluções baseadas em *hotspots Bluetooth*, seria interessante suportar a entrega de imagens de acordo com a dimensão do ecrã do dispositivo móvel. Quando fossem efectuados os pedidos aos servidores de conteúdos, seria enviada a informação relativa à dimensão do ecrã e as imagens seriam redimensionadas de forma adequada antes do envio da informação para o terminal. Desta forma, eliminar-se-ia outra limitação desta versão que é o facto de as imagens a utilizar nos conteúdos necessitarem de ser redimensionadas previamente antes do seu uso no editor de conteúdos.

Em relação à aplicação de visualização de conteúdos, foi identificada a necessidade da melhoria do suporte existente para a pré-visualização do conteúdo a ser editado na aplicação de edição. Actualmente, é feita a procura por um serviço nos terminais com suporte *Bluetooth* que se encontrem acessíveis. Esta pesquisa é efectuada em todos os dispositivos acessíveis e, dependendo do número de terminais com suporte *Bluetooth* que se encontram visíveis no local, pode demorar muito tempo. A solução passaria pela introdução do endereço MAC da máquina onde está instalada a aplicação de edição de conteúdos directamente na aplicação de visualização de conteúdos. Esta seria uma definição pessoal que ficaria guardada nas definições da aplicação. Este problema ficou parcialmente resolvido após a introdução do componente de pré-visualização de conteúdos através de NFC-IP1, uma vez que o componente de pré-visualização *Bluetooth* pode ser substituído por este último.

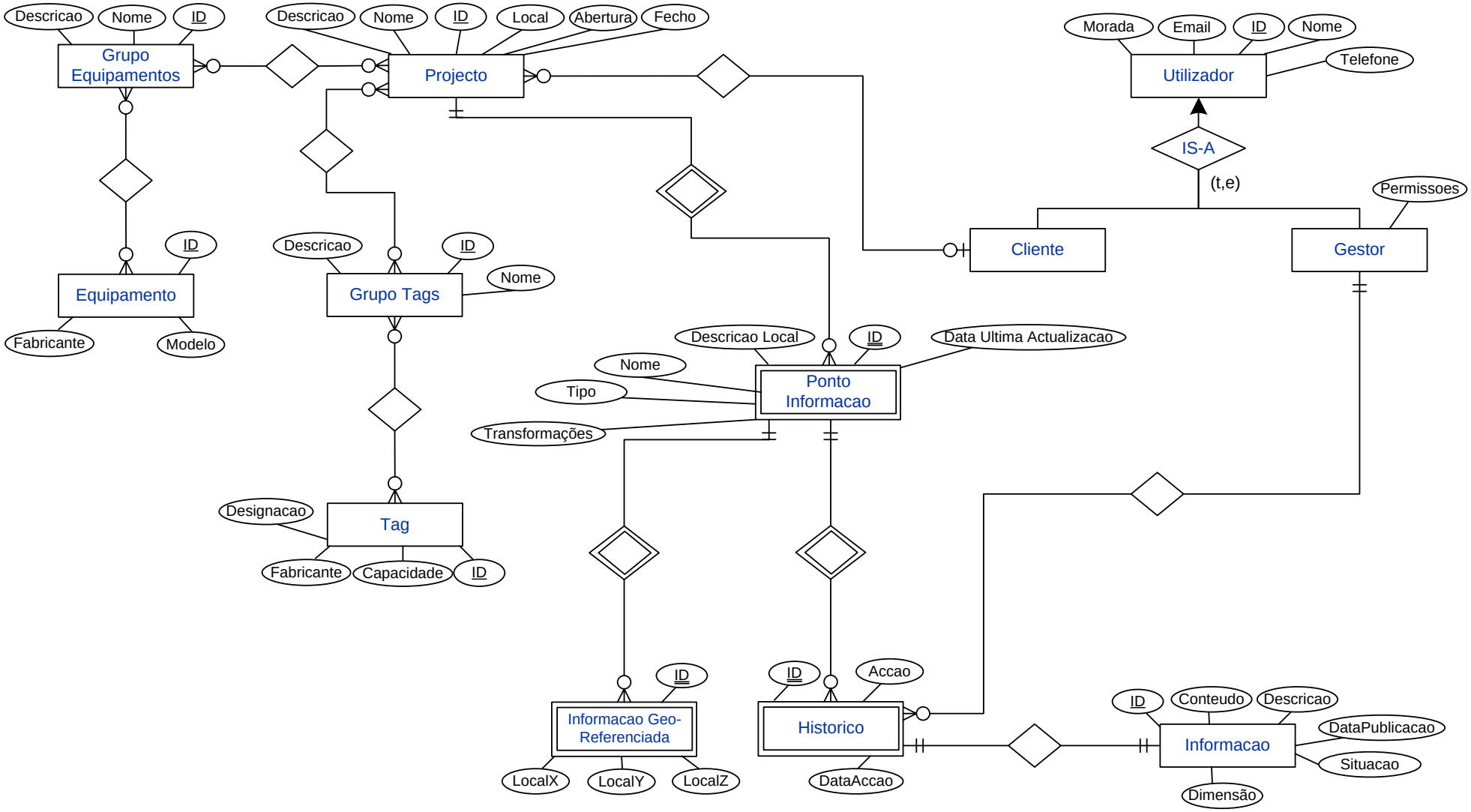
7 Anexos

7.1 Apresentação dos quatro tipos de *tags* NFC

Neste anexo é apresentado um resumo das características dos quatro tipos de *tags* especificados pelo *NFC Forum*.

- Tipo 1 [40] Ex.: *Innovision Topaz*[41].
    - Baseado no standard ISO 14443-A.
    - Suportam leitura e escrita a partir de terminais comuns.
    - Podem ser bloqueadas de modo a funcionarem apenas para leitura. Esta operação é física e irreversível.
    - Dimensões de memória:
        - Especificadas: entre 96B e 2kB.
        - Disponíveis no mercado: 96B.
- Tipo 2 [42] – Ex.: *Mifare UltraLight* (NXP)[43].
    - Baseado no standard ISO 14443-A.
    - Suportam leitura e escrita a partir de terminais comuns.
    - Possuem uma área que só pode ser escrita uma única vez.
    - Dimensões de memória:
        - Especificadas: entre 48B e 2kB.
        - Disponíveis no mercado: 48B úteis de dados, num total de 64B.
- Tipo 3 [44] – Ex.: *Sony FeliCa* RC-S880[45].
    - Baseado no standard JISX 6319-4.
    - Suportam a leitura e, dependente do próximo ponto, a escrita a partir de terminais comuns.
    - Podem ser configuradas em fábrica para suportarem apenas de leitura, ou leitura e escrita.
    - Suporta múltiplas aplicações na mesma *tag*.
    - Dimensões de memória:
        - Especificadas: o limite teórico é de 1MB por aplicação.
        - Disponíveis no mercado: entre 4kB e 9kB (valores totais)[46].
- Tipo 4 [47] – Ex.: *Mifare DESFire* [48].
    - Baseado nos standards ISO 14443-A e ISO 14443-B.
    - Suportam a leitura e, dependente do próximo ponto, a escrita a partir de terminais comuns.
    - Podem ser configuradas em fábrica para suportarem apenas de leitura, ou leitura e escrita.
    - Suporta múltiplas aplicações no mesma *tag*.
    - Dimensões de memória:
        - Especificadas: o limite teórico é de 4kB por aplicação.
        - Disponíveis no mercado: 8kB (total).

7.2 Modelo E-A da camada de persistência



8 Referências

1. NFC Forum: About NFC. (Referência de 14 Setembro, 2008) Disponível em: http://www.nfc-forum.org/aboutnfc/
2. NFC Forum: About the NFC Forum. (Referência de 14 Setembro, 2008) Disponível em: http://www.nfc-forum.org/aboutus/
3. French road test cashless city - Public Sector. In: Silicon.com. (Referência de 10 Julho, 2008) Disponível em: http://www.silicon.com/publicsector/0,3800010403,39158138,00.htm
4. Movensis: I&R. (Referência de 29 Setembro, 2008) Disponível em: http://www.movensis.com/2008/innovation/index.htm
5. Movensis, Serviços de Apoio a Comunicações, SA. (Referência de 29 Setembro, 2008) Disponível em: http://www.movensis.com/2008/index.htm
6. ECMA International: Standard ECMA 340. (2002)
7. NFC Forum: NFC Data Exchange Format. (2006)
8. NFC Forum: URI Record Type Definition. (2006)
9. NFC Forum: Record Type Definition. (2006)
10. NFC Forum: Smart Poster Record Type Definition. (2006)
11. NFC Forum: Text Record Type Definition. (2006)
12. In: WIMA 2008. (Referência de 16 Julho, 2008) Disponível em: http://www.wima.mc/mailingimg/NFC PARCOURS 3.htm
13. NXP Semiconductors: Mifare Standard 1kByte Card IC - Functional Specification. (2002)
14. IETF: Hypertext Transfer Protocol -- HTTP/1.1. (1999)
15. JSRs: Java Specification Requests - detaul JSR#135. In: The Java Community Process(SM) Program. (Referência de 15 Julho, 2008) Disponível em: http://jcp.org/en/jsr/detail?id=135
16. Device Details - Nokia 6131 NFC. (Referência de 23 Março, 2008) Disponível em: http://www.forum.nokia.com/devices/6131_NFC
17. Device Details - Nokia 6212 Classic. (Referência de 23 Setembro, 2008) Disponível em: http://www.forum.nokia.com/devices/6212_classic
18. LINQ TO SQL:.NET Language-Integrated Query for Relational Data. (Referência de 14 Agosto, 2008) Disponível em: http://msdn.microsoft.com/en-us/library/bb425822.aspx#linqtosql_topic1
19. Richter, J.: CLR via C# 2ª Edição. Microsoft Press (2006)
20. Factory method pattern. In: Wikipedia, the free encyclopedia. (Referência de 30 Setembro, 2008) Disponível em: http://en.wikipedia.org/wiki/Factory_method_pattern
21. Gamma, E.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley (2004)
22. Arygon Technologies: ACMx/APPx Series Core Module. (2005)
23. NXP Semiconductors: NFC PN531 – uC based Transmission module - Objective Short Form Specification.

(2004)

24. Provider Model Design and Specification, Part 1. (Referência de 14 Agosto, 2008) Disponível em: http://msdn.microsoft.com/en-us/library/ms972319.aspx
25. JSRs: Java Specification Requests - detail JSR#257. In: The Java Community Process(SM) Program. (Referência de 28 Junho, 2008) Disponível em: http://jcp.org/en/jsr/detail?id=257
26. Model-View-Controller. In: Wikipedia, the free encyclopedia. (Referência de 30 Setembro, 2008) Disponível em: http://en.wikipedia.org/wiki/Model-view-controller
27. NXP Semiconductors: Mifare Application Directory - Application Note. (2007)
28. NXP Semiconductors: Mifare Standard as NFC Forum Enabled Tag - Application Note. (2007)
29. Jonathan, K.: Wireless Java - Developing with J2ME 2ª Edição. Apress (2003)
30. Nokia: Nokia 6131 NFC - Technical Product Description. (2007)
31. NXP Semiconductors: Mifare Standard 4kByte Card IC - Functional Specification. (2002)
32. ISO/IEC: Information Technology - Telecommunications and information exchange between systems. (2003)
33. NXP Semiconductors: PN531/C3 - User Manual. (2005)
34. Virkus, R.: Pro J2ME Polish - Open Source Wireless Java Tools Suite. APress (2005)
35. Apache Ant - Frequently Asked Questions. In: Apache Ant. (Referência de 17 Julho, 2008) Disponível em: http://ant.apache.org/faq.html#what-is-ant
36. Introducing JSON. In: JSON. (Referência de 23 Agosto, 2008) Disponível em: http://www.json.org/
37. eXtensible Markup Language. In: World Wide Web Consortium. (Referência de 23 Agosto, 2008) Disponível em: http://www.w3.org/XML/
38. 32feet.NET - Home. (Referência de 12 Maio, 2008) Disponível em: http://www.codeplex.com/32feet/
39. Fritz, O.: Essential ASP.Net 2.0. Addison Wesley (2006)
40. NFC Forum: Type 1 Tag Operation Specification. (2007)
41. Innovision Research & Technology PLC: Topaz 13.56MHz NFC / RFID Read/Write IC. (2007)
42. NFC Forum: Type 2 Tag Operation Specification. (2007)
43. NXP Semiconductors: Contactless single-trip ticket IC - Functional specification. (2007)
44. NFC Forum: Type 3 Tag Operation Specification. (2007)
45. Sony Global - FeliCa - RC-S880. (Referência de 12 Agosto, 2008) Disponível em: http://www.sony.net/Products/felica/pdt/crd2.html
46. Sony Global - FeliCa - Product Information. (Referência de 21 Julho, 2008) Disponível em: http://www.sony.net/Products/felica/pdt/index.html#crd
47. NFC Forum: Type 4 Tag Operation Specification. (2007)
48. NXP Semiconductors: MIFARE DESFire contactless multi-application IC - Objective short data sheet. (2007)