Universidade de São Paulo
Instituto de Matemática e Estatística

Bluetooth e NFC: estudo de caso

Rodolpho Iemini Atoji

MAC499 – Trabalho de Formatura Supervisionado
Supervisor: Prof. Alfredo Goldman
Dezembro de 2010

Sumário

Sumário 1

1 Introdução 1
- 1.1 Estudo de caso . . . 2
- 1.2 Organização do texto . . . 3

2 Bluetooth – Como funciona 4
- 2.1 Visibilidade . . . 4
  - 2.1.1 Consumo de energia . . . 4
- 2.2 Descoberta de dispositivos . . . 5
  - 2.2.1 Simultaneidade . . . 5
  - 2.2.2 Determinismo e pior caso . . . 7
- 2.3 Perfis . . . 7
- 2.4 Busca de serviços . . . 8
- 2.5 Pareamento . . . 8
- 2.6 Topologia . . . 8

3 NFC – Como funciona 10
- 3.1 Modos de operação . . . 11
- 3.2 Topologia e comunicação . . . 11
- 3.3 Iniciador de comunicação secundária *Bluetooth* . . . 11
  - 3.3.1 Pareamento via Bluetooth e NFC . . . 12
- 3.4 Comparativo entre Bluetooth e NFC . . . 12

4 Estudo de caso: *marketing* de proximidade 14
- 4.1 O problema . . . 14
  - 4.1.1 Alcance . . . 14
  - 4.1.2 Não-determinismo do processo de descoberta . . . 15
  - 4.1.3 Cenário de movimentação . . . 15
  - 4.1.4 Limitação da topologia . . . 15
  - 4.1.5 Escalonamento de recursos . . . 16
- 4.2 Solução . . . 18

5 Implementação **19**
5.1 Projetos desenvolvidos . . . 19
5.2 Ambiente de execução . . . 19
5.3 JSRs e implementações . . . 20
5.3.1 JSR 82: *Java APIs for Bluetooth* . . . 20
5.3.2 JSR 257: *Contactless Communication API* . . . 21
5.4 Abordagem de problemas . . . 22
5.5 Fluxos do sistema . . . 22
5.6 Controle de concorrência e escalonamento . . . 26
5.7 Emulação e ambiente de simulação . . . 27
5.7.1 Dispositivos virtuais . . . 27
5.7.2 Deficiências do ambiente de simulação . . . 29

6 Simulações **30**
6.1 Objetivo . . . 30
6.2 Metodologia . . . 30
6.2.1 Definição do cenário de simulação . . . 31
6.2.2 Parametrização e geração de arquivos de cenários de simulação . . . 31
6.2.3 Execução do *software* de *Bluetooth marketing* no modo emulado . . . 33
6.2.4 Coleta de resultados na base de dados SQL . . . 33
6.3 Problemas e soluções de contorno . . . 33
6.4 Limitações . . . 34
6.5 Simulação 1: efeito apenas da rejeição ativa . . . 34
6.5.1 Parâmetros fixos . . . 34
6.5.2 Tempos de espera sem e com NFC . . . 35
6.6 Simulação 2: aproximação de cenário real . . . 36
6.6.1 Parâmetros . . . 37
6.6.2 Tipos de simulação e resultados . . . 38
6.7 Conclusões . . . 40

7 Parte subjetiva **42**
7.1 Desafios e frustrações . . . 42
7.2 Disciplinas mais relevantes para o trabalho . . . 43
7.3 Aplicação de conceitos estudados no curso . . . 44
7.4 Continuidade do trabalho . . . 44

A Padrões de projeto utilizados **46**
A.1 *Adapter* . . . 46
A.2 *Chain of Responsibility* . . . 47
A.3 *Factory Method* . . . 47
A.4 *Strategy* . . . 48
A.5 *Observer* . . . 49
A.6 *Singleton* . . . 50

B Utilização do *software* 52
B.1 Diretórios 52
B.2 Arquivos de configuração 53
B.2.1 Parâmetros do transmissor (`bluetooth.properties`) 53
B.2.2 Filtros (`filter.properties`) 53
B.2.3 Modo de emulação (`nfctouch.properties`) 55
B.3 Opções de linha de comando 55
B.4 Base de dados 55
B.5 Logs 55
B.6 Compilação 56
B.7 Execução 56

Referências Bibliográficas 58

Resumo

A tecnologia *Bluetooth* foi proposta inicialmente com o propósito de ser um substituto universal para a interconexão cabeada entre aparelhos eletrônicos, sejam eles quais forem.

Para atingir esse objetivo, o padrão de radiofrequência *Bluetooth* é dedicado para a comunicação a curtas distâncias (1 m a 100 m), no baixo consumo de energia e baixo custo de integração (chips de baixa complexidade e produzíveis em larga escala).

Atualmente a tecnologia *Bluetooth* equipa a grande maioria dos aparelhos celulares, *laptops* e uma série de periféricos de computador como *mouse*, teclado, fones de ouvido e outros, o que ajuda a diminuir a quantidade de fios necessária para interconectar todos esses dispositivos.

A tecnologia NFC, de *Near Field Communication* (Comunicação por Proximidade de Campo), também de radiofrequência, surge como auxiliar nesse cenário. Sua especialidade é transmitir pequenas quantidades de dados a distâncias ainda mais curtas (1 cm ou menos) e de forma segura.

Com a proliferação de dispositivos *Bluetooth*, o NFC surge como aplicação complementar, de maneira a fazer com que dispositivos *Bluetooth* se reconheçam e se autentiquem instantaneamente, em meio a uma imensa gama de dispositivos habilitados com a tecnologia que podem estar disponíveis em um mesmo ambiente.

Neste trabalho é feito um estudo de caso onde o objetivo é verificar se o NFC realmente apresenta vantagens para iniciar a comunicação *Bluetooth* em ambientes com alta concentração de dispositivos.

Para isso desenvolveu-se um sistema de *marketing* por proximidade, cujo objetivo é entregar conteúdo multimídia a dispositivos *Bluetooth* que estejam no seu raio de atuação.

Em ambientes de alta concentração de dispositivos, a principal dificuldade é escolher dentre estes quais devem receber a atenção do sistema de *marketing* por proximidade, para que ele não perca tempo com dispositivos que provavelmente não receberão o conteúdo.

Neste cenário, dispositivos *Bluetooth* com NFC, ao tocar em um leitor, anunciam ativamente sua predisposição em receber conteúdos multimídia. Isso permite que os mesmos sejam priorizados pelo sistema de *marketing*, melhorando assim a eficiência e a experiência do usuário.

A fim de verificar essa possibilidade, simulações foram realizadas com dispositivos *Bluetooth* emulados.

Capítulo 1

Introdução

O nome da tecnologia *Bluetooth* é uma referência ao rei dinamarquês Harold Blatand, cujo nome em inglês seria Harald *Bluetooth* [9]. Esse rei foi fundamental na unificação de territórios que hoje são conhecidos como Suécia, Noruega e Dinamarca. Justamente por isso ocorreu a escolha do nome *Bluetooth* pelo *Bluetooth SIG* (*Special Interest Group* – Grupo de Interesse Especial), uma vez que a tecnologia almeja a colaboração entre equipamentos de diversos setores da indústria.

O *Bluetooth* é uma tecnologia de radiofrequência que opera na faixa de frequência não-licenciada dos 2.4 GHz, também conhecida como ISM – *Industrial Scientific Medical* (Industrial Científica Médica). Nessa faixa de frequência podem operar dispositivos que não necessitem de uma legislação específica, ao contrário de canais de televisão, rádio, celular, etc.

Desde sua primeira especificação, o *Bluetooth* possui uma série de versões, onde as mais recentes especificam aumento da taxa de transferência e/ou recursos para economia de energia:

Versão	Taxa de transf.	Principais atualizações
1.1	1 Mbps	Primeira versão comercial
1.2	1 Mbps	Implementação de estratégia antiinterferência (AFH)
2.0+EDR	2.1 Mbps	Melhoria na taxa de transferência (EDR)
2.1+EDR	2.1 Mbps	Processo de pareamento simples (SSP)
3.0+HS	24 Mbps	Uso de canal WiFi (IEEE 802.11) para transferência
4.0	1 Mbps	Versão de consumo de energia altamente reduzido

Tabela 1.1: Versões da especificação *Bluetooth*

Como adicional às especificações, cada dispositivo que incorpora um controlador *Bluetooth* pode ser enquadrado em cada uma das classes vistas na tabela 1.2, que especificam o alcance do mesmo.

Ainda de acordo com a tabela 1.2, quanto maior o alcance, maior a potência necessária para fazer o transmissor atingir a distância desejada. Por essa razão dispositivos portáteis normalmente são de classe 2 ou 3.

Para que dispositivos *Bluetooth* se comuniquem entre si, há uma etapa opcional do processo denominada **pareamento**. Nesse processo, no ato da comunicação entre dois dispositivos é especificada uma **chave de pareamento** para que ambos se autentiquem e a conexão seja

Classe	Distância	Potência
3	até 1 m	1 mW
2	até 10 m	2.5 mW
1	até 100 m	100 mW

Tabela 1.2: Classes *Bluetooth*

criptografada.

O processo de pareamento usual normalmente transcorre pelas seguintes etapas:

1. **Descoberta:** o dispositivo-alvo deve ser descoberto pelo dispositivo que deseja iniciar comunicação com o mesmo;

2. **Busca de serviço:** ao ser questionado sobre os serviços que provê, o dispositivo-alvo requisita uma autenticação;

3. **Chave de pareamento:** uma chave de pareamento deve ser inserida em ambos os dispositivos que desejam se comunicar. Normalmente é uma chave numérica que deve ser digitada ao se transferir dados para um dispositivo pela primeira vez.

A partir da versão 2.1 da especificação *Bluetooth* foi incluído o mecanismo SSP – *Secure Simple Pairing* (Pareamento Seguro Simples) – que aceita alguns métodos diferentes para atingir esse objetivo. Um dos métodos é o OOB – *Out Of Band* (Fora Da Banda) – que especifica que algum mecanismo diferente do *Bluetooth* pode ser utilizado para o pareamento.

Neste cenário é que se inserem as tecnologias de proximidade, conhecidas pela sigla NFC (*Near Field Communication* – Comunicação por Proximidade de Campo). Dois dispositivos *Bluetooth*, dotados também de tecnologia NFC, podem realizar o pareamento pelo simples contato entre um e outro, reduzindo o tempo normalmente gasto pelo processo usual, já que os processos de descoberta e busca de serviço são eliminados.

1.1 Estudo de caso

A fim de verificar o benefício da eliminação dos processos de descoberta e busca de serviço, foi implementado um sistema de *Bluetooth marketing* ou *marketing* de proximidade.

Este sistema consiste no envio de conteúdos multimídia a dispositivos habilitados com *Bluetooth* que estejam na sua área de atuação.

Ao invés de ser encontrado, o dispositivo Bluetooth anuncia ativamente sua presença, por meio da tecnologia NFC. Uma vez que o NFC busca auxiliar a transferência de dados via *Bluetooth*, o leitor localiza-se dentro da área de atuação do mesmo.

Para verificar e simular a solução, um sistema de *Bluetooth marketing* foi implementado em Java (JSR 82). Esse sistema continuamente descobre novos dispositivos *Bluetooth* na vizinhança e envia conteúdos multimídia àqueles que aceitarem o pedido de conexão. Os dispositivos que fizerem uso da tecnologia NFC são priorizados e instantaneamente servidos.

1.2 Organização do texto

Nos **capítulos 2** e **3** são apresentadas as tecnologias *Bluetooth* e NFC, assim como um breve comparativo entre as duas.

O **capítulo 4** detalha o funcionamento de um sistema de *marketing* por proximidade, os problemas enfrentados pelo mesmo para a entrega de conteúdos multimídia e as soluções propostas.

No **capítulo 5** é feita a descrição do sistema de *Bluetooth marketing* implementado, de acordo com as soluções propostas no capítulo anterior. Essa descrição leva em conta aspectos técnicos do sistema e fluxos de execução.

A simulação da interação entre *Bluetooth* e NFC no sistema de transmissão de conteúdos multimídia é tratada no **capítulo 6**, que mostra os resultados de diversas execuções com parâmetros variados do sistema de *marketing* sobre dispositivos emulados.

O **capítulo 7** apresenta uma visão subjetiva do trabalho de conclusão de curso, abordando disciplinas relevantes para a execução do trabalho, aplicação de conceitos estudados entre outros.

Por fim, os **apêndices A** e **B** mostram, respectivamente, padrões de projeto utilizados na implementação do sistema de *Bluetooth Marketing* e instruções para execução do mesmo.

Capítulo 2

Bluetooth – Como funciona

A seguir serão apresentados alguns aspectos técnicos sobre a tecnologia *Bluetooth*, especialmente no que diz respeito ao processo de descoberta de dispositivos, busca de serviços e pareamento.

2.1 Visibilidade

Em termos de visibilidade para outros dispositivos *Bluetooth*, um equipamento pode estar nos seguintes estados [10]:

- **Visível:** o dispositivo pode ser descoberto por qualquer outro que realize um processo de descoberta na sua vizinhança e permanece nesse estado por tempo indeterminado;
- **Visível por tempo limitado:** uma variação do estado acima, permite que dispositivos fiquem visíveis apenas pelo tempo necessário para que sejam descobertos;
- **Oculto:** o dispositivo não responde a processos de descoberta realizados pelos seus vizinhos, mas aceita pedidos de conexão com aqueles que já o conhecem.

Estando em um dos modos *visível* ou *visível por tempo limitado*, o dispositivo *Bluetooth* monitora requisições de descoberta em um canal exclusivo para essa finalidade.

Cada canal possui uma série de frequências onde podem ser disparadas requisições de descoberta. O dispositivo no estado visível itera por cada uma dessas frequências, respondendo a requisições de descoberta em cada uma, se for o caso.

2.1.1 Consumo de energia

Vale observar que cada um dos estados é determinante para o consumo de energia do aparelho habilitado com *Bluetooth*.

O modo *visível* é o que mais gasta energia, pelo fato de o sistema estar o tempo todo iterando sobre as frequências de descoberta, a fim de responder a eventuais requisições.

Já o modo *visível por tempo determinado* itera pelas frequências de descoberta por um tempo que normalmente fica entre um a cinco minutos, prazo após o qual entra automaticamente no modo *oculto*, gastando portanto energia para o processo de descoberta apenas nesse intervalo de visibilidade.

No modo *oculto* não há gasto de energia para o processo de descoberta, sendo portanto o melhor estado para comunicação entre dispositivos que já se conheçam previamente.

2.2 Descoberta de dispositivos

A **descoberta de dispositivos** é um processo o qual qualquer dispositivo *Bluetooth* pode iniciar, de maneira a detectar outros dispositivos habilitados com *Bluetooth* no seu raio de alcance.

Para essa finalidade, o dispositivo que inicia o processo de descoberta envia uma mensagem de *broadcast* (difusão) no canal de descoberta de dispositivos e aguarda pela resposta daqueles que estiverem visíveis e respondam à requisição.

Essa mensagem de *broadcast* é acompanhada de um dos dois possíveis códigos de acesso, GIAC ou LIAC [10]:

- **GIAC (General Inquiry Access Code):** ou Código de Acesso para Descoberta em Geral, especifica que todos os dispositivos que estejam no modo visível permanente devem responder a essa requisição;
- **LIAC (Limited Inquiry Access Code):** ou Código de Acesso para Descoberta Limitada, é utilizado para dispositivos que estejam visíveis apenas temporariamente.

A necessidade do uso de dois códigos de acesso se justifica à medida em que se deseja localizar um dispositivo muito próximo, em meio a muitos dispositivos habilitados com *Bluetooth*. Quando uma descoberta é realizada com o LIAC, uma quantidade bastante reduzida de dispositivos responde à requisição, passando rapidamente para o modo oculto.

A resposta ao pedido de descoberta inclui principalmente um identificador do dispositivo, dado pelo endereço de rede físico do *Bluetooth*, que é o *MAC address – Media Access Control address* (Endereço de Controle de Acesso ao Meio) – similar aos endereços utilizados nas controladoras de rede Ethernet.

Uma vez de posse desse identificador, um dispositivo pode dar início a processos de requisição de conexão ao dispositivo descoberto.

2.2.1 Simultaneidade

O processo de descoberta não pressupõe nenhum controle de concorrência, tampouco topologia da rede.

Ao mesmo tempo em que um dispositivo deflagra um processo de descoberta, outros dispositivos podem também estar realizando os seus processos.

Dessa maneira, um dispositivo que inicia o processo de descoberta pode estar simultaneamente descobrindo outros aparelhos e respondendo a requisições de descobertas de outros aparelhos (inclusive os que não tenham ainda sido descobertos).

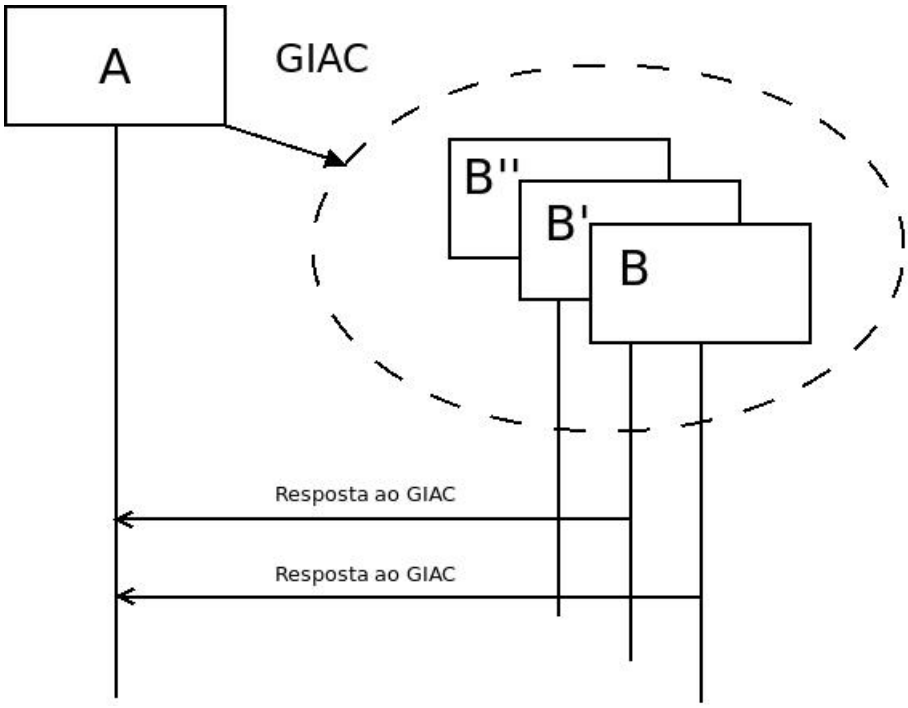


Figura 2.1: Descoberta geral (GIAC), onde B" está oculto, B' está visível por tempo limitado, e B em modo visível.)

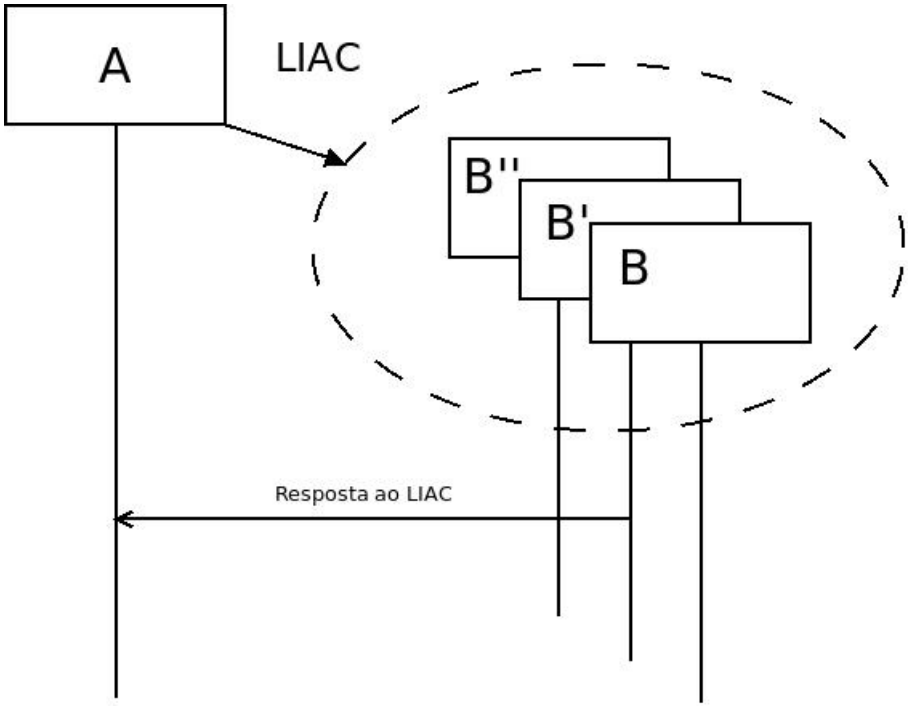


Figura 2.2: Descoberta limitada (LIAC), onde B" está oculto, B' está visível por tempo limitado e B em modo visível.

2.2.2 Determinismo e pior caso

O processo de descoberta de dispositivos é **não-determinístico**.

Isso significa que numa mesma área de atuação, para os mesmos dispositivos, sucessivos processos de descoberta deflagrados por um dispositivo podem devolver sucessivos conjuntos diferentes de dispositivos encontrados.

Isso se deve principalmente a dois fatos: resposta pseudoaleatória na frequência de descoberta e limitação do tempo no processo de descoberta.

A resposta pseudoaleatória ocorre quando um dispositivo visível responde a uma requisição de descoberta. Para evitar colisão com outros dispositivos, espera-se um tempo pseudoaleatório para responder [3]. Em ambientes onde há um número muito grande de dispositivos, podem ocorrer sucessivas colisões (interferências) e com isso os dispositivos envolvidos podem não ser descobertos caso o tempo de descoberta não seja o suficiente para que se saia dessa situação.

O tempo-limite do processo de descoberta é variável para cada dispositivo. Um período de transmissão/recepção (TX/RX) leva 1250 ms, normalmente os dispositivos executam 8 desses períodos, totalizando 10 s para um intervalo de busca recomendado para que se descubram todos os dispositivos na vizinhança [10], em condições livres de interferência.

2.3 Perfis

Além de especificar *hardware*, protocolos de comunicação e outros componentes de baixo nível, a especificação *Bluetooth* também formaliza serviços de alto nível, denominados **perfis**.

Os perfis determinam um padrão para diversas atividades que podem ser realizadas por meio de dispositivos que possuam *Bluetooth* integrado, como por exemplo troca de arquivos, transmissão de voz, impressão, conexão com a Internet, etc.

Sigla	Perfil	Descrição
AD2P	Advanced Audio Distribution Profile	Transmissão de som estéreo
BPP	Basic Printing Profile	Impressão
DUN	Dial-Up Networking	Conexão com a Internet
FAX	FAX Profile	Envio de FAX
FTP	File Transfer Profile	Transferência de arquivos
HID	Human Interface Device	Dispositivos como mouse, teclado, etc
OPP	Object Push Profile	Envio direto de dados

Tabela 2.1: Alguns perfis *Bluetooth*

Cada perfil possui um identificador exclusivo. Com base no mesmo, é possível saber quais serviços cada dispositivo *Bluetooth* oferece.

Essa característica permite que dispositivos *Bluetooth* de implementações completamente distintas possam interagir imediatamente, utilizando a especificação do perfil como interface de comunicação.

2.4 Busca de serviços

A tecnologia *Bluetooth* especifica um protocolo denominado SDP (*Service Discovery Protocol* – Protocolo de Descoberta de Serviços), que pode ser executado em um dispositivo descoberto após o respectivo processo.

O protocolo funciona no esquema cliente/servidor, onde o cliente é o dispositivo que inicia o SDP e o servidor é o dispositivo-alvo. No caso do SDP, todos os dispositivos podem ser clientes ou servidores ao mesmo tempo, depende apenas de qual lado inicia o processo [10].

O dispositivo-alvo, ao receber uma requisição de SDP, devolve uma lista de registros denominados *service records* – registros de serviço.

Cada um desses registros está associado a um perfil suportado pelo dispositivo-alvo, bem como contém informações específicas de como deve ser feita a conexão com o mesmo a fim de que o serviço seja realizado. Essas informações são específicas de cada sistema que implementa o *Bluetooth* (como qual canal de radiofrequência deve ser utilizado, protocolo, etc).

2.5 Pareamento

O pareamento (*pairing*) é um processo opcional mas que pode ser deflagrado ainda durante a execução do SDP.

Esse recurso garante a autenticação dos dispositivos que estão se comunicando por meio de uma chave numérica pré-combinada, por exemplo.

Caso um dispositivo-alvo decida que o pareamento é obrigatório, ele pode requerer o pareamento antes ou depois do SDP, momento no qual ele pergunta qual a chave pré-combinada. O dispositivo que iniciou o SDP entra a chave e então o dispositivo-alvo faz a autenticação da mesma, permitindo a comunicação em caso de sucesso.

Normalmente a chave para essa tarefa é muito simples e pode ser quebrada rapidamente, dependendo do tamanho [8]. Portanto, é recomendável que dispositivos que utilizem esse recurso utilizem chaves bem maiores que os quatro dígitos usuais.

O método de pareamento percorre os seguintes estágios [10] :

1. **Descoberta de dispositivos:** uma consulta aos dispositivos presentes na área de atuação é realizada;
2. **Conexão:** uma conexão é criada com o dispositivo descoberto pelo processo anterior;
3. **Procedimentos de segurança:** trocas de chaves públicas, capacidades de comunicação, etc;
4. **Autenticação:** utilização de um dos métodos disponíveis, como entrada de senha, comparação numérica, etc.

2.6 Topologia

A tecnologia *Bluetooth* permite dois tipos de conexão: ponto-a-ponto (*peer-to-peer*) e ponto-a-multiponto (*point-to-multipoint*) [2].

A conexão ponto-a-ponto permite a realização da topologia *piconet* onde um dos dispositivos é o mestre e outros dispositivos que compartilham o mesmo canal físico de comunicação são os escravos. Para cada *piconet* podem existir até sete dispositivos *Bluetooth*.

Outra topologia possível é a *scatternet*, que é a agregação de várias *piconets*. Nesse caso, o mestre de uma *piconet* pode ser o escravo em uma *scatternet*. O compartilhamento do meio físico é feito apenas entre as *piconets*, e não entre toda a *scatternet*.

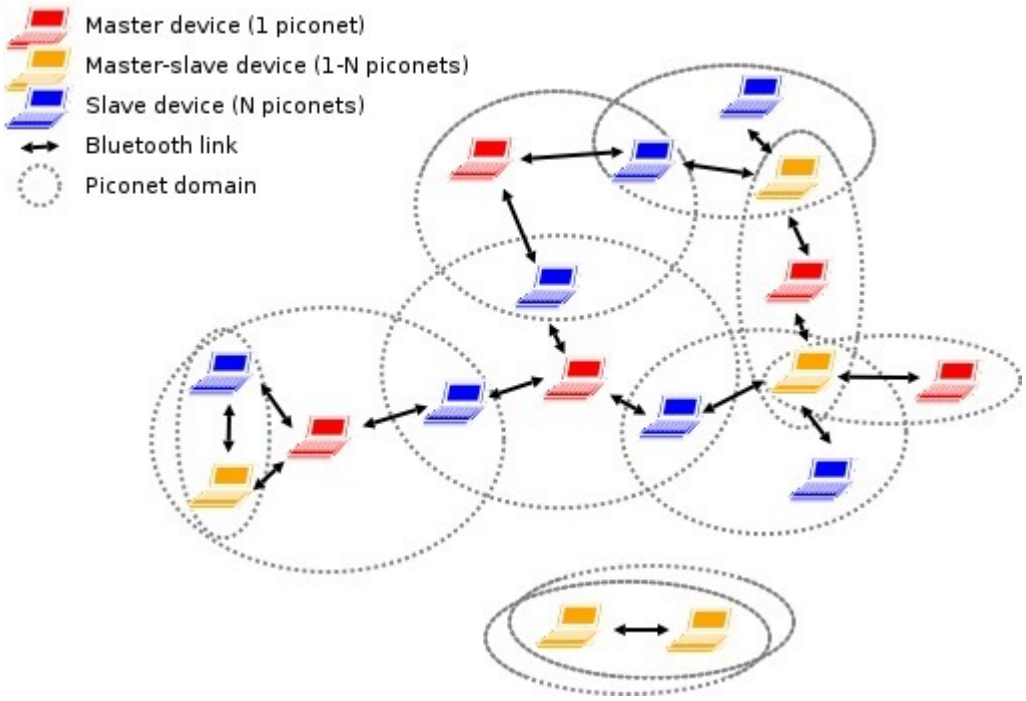


Figura 2.3: Topologias *Bluetooth* (Wikimedia commons)

Capítulo 3

NFC – Como funciona

NFC é a sigla para *Near Field Communication* – Comunicação por Proximidade de Campo. O NFC também é uma tecnologia de radiofrequência, mas diferencia-se pela distância de operação, normalmente de 0 a 20 cm entre os dispositivos.

A ideia do NFC é tornar a comunicação entre dispositivos mais simples: basta aproximá-los para que isso ocorra, eliminando a necessidade de procedimentos eventualmente complexos.

Pelo fato da distância de operação ser curta, a comunicação é inerentemente segura no que diz respeito a tentativas de interceptação. Caso haja necessidade de coibir esse tipo de ação, as medidas usuais de segurança devem ser implementadas no protocolo de rede e/ou aplicação [5].

O NFC atua na faixa dos 13.56 MHz – frequência também que é livre de restrições na maioria dos países – com velocidades típicas entre 106 e 424 kbit/s.

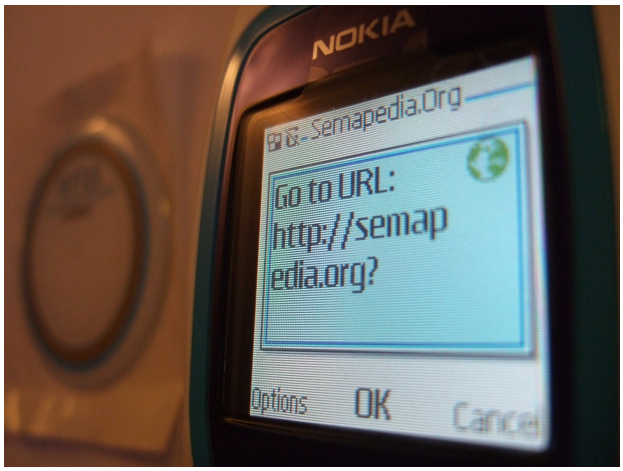


Figura 3.1: Celular lendo uma etiqueta RFID (http://www.flickr.com/photos/74845103@N00/344484054/)

A tecnologia pode ser utilizada essencialmente para dois propósitos:

- **Transmissão de pequena quantidade de dados:** como a ideia do NFC não é prover grandes velocidades, mas sim uma comunicação segura a curta distância, o mesmo não é adequado para transmissão de grandes volumes de dados. Dessa forma, dados como informações de pagamento, cartões de visita, autenticação, são os mais indicados para transferências exclusivamente pelo NFC;

- **Iniciador de comunicação secundária:** fazendo com que dois dispositivos se reconheçam pela simples aproximação, o NFC pode ser utilizado para disparar um canal de comunicação secundário, como *Bluetooth* ou WiFi.

A especificação NFC prevê ainda que a tecnologia seja compatível com dispositivos RFID (*Radiofrequency IDentification* – IDentificação por Radiofrequência), que operam na mesma

frequência e mesmos modos de operação (especialmente os passivos). Dessa maneira muitos dispositivos habilitados com NFC são capazes de emular cartões com a tecnologia RFID ou mesmo *tags* (etiquetas) RFID.

3.1 Modos de operação

Dois modos de operação estão disponíveis na tecnolgia: ativo e passivo [5].

No modo de comunicação **ativo** ambos os dispositivos são capazes de gerar seus sinais de radiofrequência para a comunicação. Normalmente são dispositivos dotados de fonte de energia própria, como bateria ou energia elétrica convencional. Os leitores NFC são exemplos de dispositivos ativos.

Já no modo de comunicação **passivo** apenas um dispositivo (denominado *iniciador*) gera o sinal de radiofrequência para comunicação. Nesse caso, um campo eletromagnético é gerado, e esse campo induz uma corrente elétrica no dispositivo passivo, que faz com que o mesmo tenha energia para funcionar pelo que tempo em que estiver sob ação do campo referido. Cartões NFC são exemplos de dispositivos passivos.

Figura 3.2: Etiqueta RFID passiva (http://www.flickr.com/photos/74845103@N00/415981279/)

3.2 Topologia e comunicação

Na tecnologia, o número de participantes na comunicação é limitado a dois, sendo portando um protocolo *peer-to-peer*.

Pelo fato dos dispositivos compartilharem uma única banda de radiofrequência, a comunicação é *half-duplex*, ou seja, apenas um dispositivo transmite dados por vez.

A estratégia adotada para esse tipo de comunicação é conhecida como "ouça antes de se comunicar" [5], onde o dispositivo participante na comunicação deve primeiro ver se não tem ninguém transmitindo dados no momento e só então iniciar a sua comunicação.

3.3 Iniciador de comunicação secundária *Bluetooth*

A partir da versão 2.1 o *Bluetooth* passa a suportar o SSP – *Secure Simple Pairing* (Pareamento Seguro Simples) – onde métodos alternativos ao *Bluetooth* podem ser utilizados para executar o pareamento, a fim de agilizar e/ou adicionar mais segurança a esse procedimento.

Em dispositivos *Bluetooth* também habilitados com NFC, o pareamento ocorre em uma fração do tempo normalmente necessário para que o procedimento ocorra exclusivamente por

Bluetooth. Enquanto no *Bluetooth* o tempo para para estabelecimento de uma conexão leva entre 6 a 10 segundos (*set-up time*), no NFC ele ocorre em até 0.1 segundo.

O processo de SSP percorre os seguintes estágios [10]:

1. **Pareamento:** descoberta e autenticação via NFC;
2. **Descoberta de dispositivos:** informações sobre o dispositivo *Bluetooth* são enviadas diretamente via NFC;
3. **Conexão:** por meio do endereço MAC *Bluetooth* enviado via NFC, uma conexão *Bluetooth* é estabelecida;
4. **Procedimentos de segurança:** trocas de chaves públicas, capacidades de comunicação, etc;
5. **Autenticação:** informação de autenticação enviada diretamente via NFC.

3.3.1 Pareamento via Bluetooth e NFC

A tabela 3.1 mostra um comparativo entre os dois processos de pareamento:

Processos	Bluetooth	NFC
Canal de pareamento	Bluetooth	NFC
Descoberta	*Device Inquiry*	Dados de identificação enviados diretamente
Conexão	Utiliza resultado do *Inquiry*	Endereço MAC enviado diretamente
Segurança	Troca de chaves públicas, capacidades de comunicação, etc	
Autenticação	Comparação numérica, senha, etc.	Autenticação por informação transportada pelo NFC

Tabela 3.1: Comparativo entre pareamento via *Bluetooth* e NFC

3.4 Comparativo entre Bluetooth e NFC

A tabela 3.2 confronta as principais características de cada uma das tecnologias.

É fácil ver que ambas são completamente diferentes em todas as suas especificações, o que mostra que a interação entre as duas é complementar e não concorrente.

	Bluetooth	NFC
Frequência	2.4 GHz	13.56 MHz
Velocidade	até 24 Mbps	até 424 kbps
Máximo de dispositivos	7/piconet	2
Modo de comunicação	Full-duplex	Half-duplex
Set-up time	de 6 a 30 s	em torno de 0,1 s
Distância de operação	até 100 m	até 20 cm

Tabela 3.2: Comparativo entre *Bluetooth* e NFC

Capítulo 4

Estudo de caso: *marketing* de proximidade

Dado o alcance relativamente limitado do *Bluetooth*, o mesmo pode ser utilizado para realizar ações de propaganda que atraiam um público-alvo para determinada localidade. Este tipo de ação é denominado "*Bluetooth marketing*" ou "*marketing* de proximidade".

Normalmente essas ações transcorrem da seguinte forma, considerando uma pessoa que possua um celular habilitado com *Bluetooth*:

1. Ao se aproximar da área de atuação *Bluetooth*, o usuário é convidado a aceitar um conteúdo multimídia (imagem, vídeo, etc);
2. Aceitando a recepção do conteúdo, o usuário pode recebê-lo livremente por toda a área de atuação;
3. Recebido o conteúdo, este provavelmente direcionará o usuário para alguma loja próxima, por meio de um oferecimento exclusivo no conteúdo multimídia.

4.1 O problema

Apesar de a aplicação da tecnologia seguir esses simples passos, na prática uma série de problemas surgem, que podem acabar tornando o sistema impraticável caso não sejam tomados alguns cuidados, bem como sejam feitas algumas otimizações na utilização de recursos disponíveis.

4.1.1 Alcance

Conforme listado na tabela 1.2, o *Bluetooth* possui uma série de alcances possíveis, limitado a 100 m.

No entanto, na prática é possível verificar que esses alcances só ocorrem em condições ideais, como por exemplo: livre de outros dispositivos atuando na banda de 2.4 GHz (WiFi, aparelho de microondas, telefones sem fio), sem barreiras físicas (paredes, móveis), etc. Esses fatores

provocam tanto interferências no sinal *Bluetooth* como atenuação do mesmo, impedindo que o mesmo alcance a distância ideal projetada.

Outro fator é que normalmente dispositivos móveis dotados de suprimento de energia limitada (celulares, *palmtops*, etc) não implementam o *Bluetooth* Classe 1, por exemplo, já que o consumo de energia do mesmo é incompatível com a duração da bateria do aparelho em questão. Por isso, mesmo que a ação de *marketing* use aparelhos Classe 1 para difusão de conteúdos, não há garantias de que os dispositivos-alvo serão capazes de se comunicarem na mesma distância.

4.1.2 Não-determinismo do processo de descoberta

Em áreas com alta concentração de pessoas, a probabilidade de que hajam dispositivos *Bluetooth* ativos é muito maior.

Caso o número de pessoas com *Bluetooth* ativo seja muito grande, uma execução do processo de descoberta pode não trazer como resultado todos os dispositivos ali presentes, conforme relatado na seção 2.2.2.

Dessa maneira, mesmo que um usuário deseje receber o conteúdo, ele passará despercebido pelo sistema de *marketing*, o que é algo altamente indesejável.

4.1.3 Cenário de movimentação

Uma regra geral, especialmente na comunicação sem fio é: quanto mais perto os transmissores/receptores estiverem, melhor a qualidade do sinal e da transmissão de dados.

Esse é um aspecto importante na aplicação da tecnologia, uma vez que a distância do usuário ao dispositivo de *marketing* influencia diretamente na quantidade de erros e na velocidade de transmissão.

Usuários que estejam parados na área de atuação do *Bluetooth* são mais propensos a serem rapidamente encontrados, bem como devem receber o conteúdo mais rapidamente. Isso não é necessariamente verdade se o usuário estiver parado nos limites da área de atuação do *Bluetooth*, conforme apresentado na seção 4.1.1.

Já usuários que se movimentem rapidamente pela área de atuação *Bluetooth*, (levando em conta que a mesma é limitada e o alcance máximo de 100 m é abreviado por obstáculos, interferências, etc) muito provavelmente não permanecerão na mesma tempo o suficiente para que ocorram os processos de descoberta, busca de serviços e envio de conteúdo (seções 2.2 e 2.4).

4.1.4 Limitação da topologia

Como visto na seção 2.6, cada *piconet* suporta um máximo de 7 dispositivos comunicando simultaneamente.

Isso restringe, portanto, o número de dispositivos que podem ser alcançados por vez.

Um sistema de *marketing* por proximidade pode contar com uma série de transmissores *Bluetooth*, o que ameniza essa limitação. Considerando um ambiente onde há n dispositivos *Bluetooth* ativos e um sistema de transmissão de conteúdos que possui t transmissores e cada

um suporta até 7 conexões simultâneas, temos que $t = n/7$ para os n dispositivos serem servidos simultaneamente.

Uma conta simples revela que em um cenário com 255 dispositivos ativos, serão necessários 37 transmissores *Bluetooth* para servir todos simultaneamente.

Essa normalmente não é a realidade de nenhum sistema de *marketing* por proximidade, seja por fatores técnicos ou econômicos. Entre os fatores técnicos importantes são que sistemas operacionais que gerenciam esses transmissores normalmente impõem limite no número de transmissores suportados, e quanto maior o número de transmissores, maior o número de operações de E/S (entrada/saída).

4.1.5 Escalonamento de recursos

Alguns fatores podem afetar diretamente o desempenho de um sistema de *marketing* por proximidade: erros na busca de serviços, erros na transmissão de conteúdos, número de dispositivos ativos na área de atuação, movimentação, versão do *Bluetooth*, etc.

Em uma ponta do cenário temos um sistema de *marketing* com um número de transmissores capaz de servir com folga todos os usuários na área de atuação. Na outra ponta, temos um sistema onde o número de antenas é insuficiente para cobrir todos os usuários simultaneamente, o que é a situação mais comum.

Se em um processo de descoberta de dispositivos n são encontrados, isso não significa que todos tem a mesma capacidade de receber conteúdos com sucesso.

Levando em conta a limitação da quantidade de transmissores *Bluetooth* do sistema de *marketing*, vários fatores devem ser bem-escalonados de maneira que o aproveitamento dos transmissores seja máximo, dessa forma garantindo o maior número de sucessos (aceitação completa de conteúdo) por períodos de tempo.

Transmissores com ocupação abaixo da capacidade, ou concentrando-se em dispositivos que possuem alta chance de não receber os conteúdos (passageiros, alto índice de erros, etc), farão com que dispositivos aptos a receber os conteúdos ou demorem muito a fazê-lo ou sejam completamente ignorados.

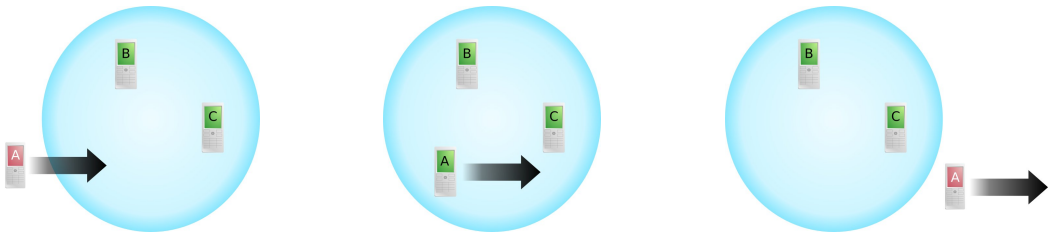
A seguir, exemplos de alguns fatores a serem considerados no escalonamento de recursos.

Usuário passageiro

A fim de determinar quais são os dispositivos-alvo para envio de conteúdos multimídia, um sistema de *Bluetooth marketing* deve realizar uma série de processos de descoberta em sequência.

Como resultado da execução de cada processo, é obtida uma relação de dispositivos descobertos. Dispositivos que já foram descobertos podem responder novamente a processos de descoberta, o que significa que os mesmos ainda estão sob a área de atuação do sistema de *marketing* pro proximidade.

Supondo que em uma sequência de dez execuções do processo de descoberta um dado dispositivo *Bluetooth* só apareça na primeira execução, pode-se considerar que o mesmo tenha estado na área de atuação apenas de passagem, devendo, portanto, ser descartado ou ser menos priorizado em relação a outros que apareçam com frequência em sucessivas execuções do processo.



(a) t1 **(b)** t2 **(c)** t3

Figura 4.1: Sequência de três execuções do processo de descoberta: o dispositivo A é encontrado apenas na segunda execução, enquanto B e C são encontrados nas três execuções (a área azul é a área de atuação do *Bluetooth*)

Erros de busca de serviço

Após ser detectado pelo processo de descoberta, deve-se realizar a busca de serviços no dispositivo-alvo, a fim de determinar se o mesmo tem a capacidade de receber conteúdos multimídia.

No entanto, uma série de fatores podem fazer com que o processo falhe: interrupção do canal de comunicação (interferência), movimentação para fora da área de atuação, bloqueio pelo dispositivo a requisições de busca de serviço, etc.

Nesses casos, não é possível enviar conteúdos para esses dispositivos até que o processo de busca de serviços tenha sucesso. Portanto, aqueles que passaram com sucesso pelo processo devem ser priorizados no envio de conteúdos.

Erros na transmissão de conteúdos

Conforme fatores citados em 4.1.1, uma transmissão pode pode apresentar uma série de erros.

Erros que culminem na interrupção do canal de transferência, forçam o sistema de *marketing* a abrir uma nova conexão e a retransmitir o conteúdo do início, uma vez que não é possível retomar transferências parciais.

Dessa forma, dispositivos que sejam mais confiáveis nesse aspecto (isto é, apresentam menor taxa de erros de transmissão) podem ser priorizados frente aos outros, uma vez que a transmissão de conteúdos é responsável pelo maior tempo de ocupação dos transmissores *Bluetooth*.

Transmissões que apresentem erros normalmente ficam sujeitas a *timeouts* (tempo de expiração), períodos nos quais o transmissor *Bluetooth* fica ocioso, desperdiçando portanto a capacidade de envio do sistema.

Aceitação

Um usuário tem a opção de receber ou não um conteúdo via *Bluetooth*.

Essa condição sozinha gera uma série de situações a serem consideradas: o usuário pode aceitar o recebimento, rejeitar ou simplesmente ignorar.

Como o sistema de *marketing* em geral pode estar programado para enviar uma série de conteúdos, uma série de permutações das situações acima é possível.

Nesse caso, a política do sistema deve decidir quais das situações deve priorizar. Em determinadas situações pode ser mais interessante que todos os dispositivos recebam ao menos um conteúdo, enquanto que em outros casos o interessante é que um dispositivo receba o máximo de conteúdos possível. Nesse último caso, o índice de aceitação deve ser considerado a fim de escolher corretamente os dispositivos-alvo passíveis de serem anunciados.

4.2 Solução

É evidente que um fator complicador do sistema de *marketing* de proximidade é descobrir quem está ao seu redor, bem como classificá-los de uma maneira em que a eficiência do sistema seja a maior possível.

Conforme apresentado na seção 3.3, o NFC é uma alternativa que permite que uma conexão *Bluetooth* seja realizada entre dois aparelhos de maneira rápida, bastando que ocorra o toque entre eles, havendo assim reconhecimento mútuo.

Dessa maneira, o NFC pode ser utilizado para priorizar ao máximo os usuários que desejem receber conteúdo via *Bluetooth*. Com a eliminação do processo de descoberta e busca de serviços, um dispositivo habilitado com NFC pode ser imediatamente reconhecido pelo sistema de *marketing*.

Considerando que os leitores NFC devem estar em áreas de cobertura *Bluetooth* as melhores possíveis, um dispositivo que faça a opção de usar NFC estará sujeito à melhor qualidade de serviço, bem como poderá receber os conteúdos o quanto antes (antes mesmo até de algum usuário que tenha chegado antes do mesmo ao local e não tenha feito a opção pelo uso de NFC).

Capítulo 5

Implementação

A implementação objetiva uma solução para os problemas relatados em 4.1.

No decorrer deste capítulo o termo "adaptador *Bluetooth*" (*Bluetooth dongle*) será utilizado para denotar qualquer tipo de controlador *Bluetooth* ligado a um computador, seja por portas e *hubs* USB disponíveis, *slots* de I/O ou integrados.

O sistema é implementado em Java e possui dois modos de operação:

- **Modo *Bluetooth*:** neste modo o *software* opera utilizando o *hardware* (um adaptador ou mais) *Bluetooth* reconhecido pelo sistema operacional;
- **Modo de emulação:** este modo permite que testes unitários e simulações possam ser realizadas sem a necessidade de adaptadores *Bluetooth*.

5.1 Projetos desenvolvidos

Foram desenvolvidos três projetos:

- **Bluetooth/NFC:** *software* principal realizando a função de *marketing* de proximidade (será referenciado apenas como "*software*" ou "programa" neste capítulo);
- **Bluetooth/NFC Emulator:** implementa dispositivos *Bluetooth* e NFC virtuais, permitindo que o *software* principal possa interagir com um número arbitrário dos mesmos;
- **JSR 257 RMI:** implementação restrita da JSR 257, simulando um leitor NFC onde os toques são realizados por chamadas de métodos via RMI.

5.2 Ambiente de execução

Apesar de ser desenvolvido em Java, o programa apresenta algumas restrições no que se refere ao ambiente de execução.

Isso se deve especialmente às diferenças entre as *stacks* (pilhas de protocolo) *Bluetooth* disponíveis em diferentes sistemas operacionais.

O sistema operacional Windows, especificamente, restringe o uso máximo de adaptadores *Bluetooth* a apenas um. Apesar de o *software* implementado ser capaz de lidar com essa situação, a plataforma mostra-se desinteressante para lidar com problemas de grande volume.

Por essa razão o programa implementado deve ser preferencialmente executado em um sistema Linux a fim de que toda sua capacidade possa ser aproveitada (exceto no modo de emulação).

Sistema operacional	Ubuntu Linux 10.4 (kernel 2.6.32-24-generic i686)
Stack Bluetooth	BlueZ 4.60-0ubuntu8
Versão JavaSE	Sun 1.6.0_20-b02
Adaptadores *Bluetooth*	Dispositivos USB equipados com *chipsets* CSR *Bluetooth* 2.1+EDR Classe 1

Tabela 5.1: Ambiente utilizado para o desenvolvimento

5.3 JSRs e implementações

JSRs – *Java Specification Requests* (Requisição de Especificações Java) são documentos que descrevem recursos atuais e futuros da plataforma Java.

No caso do *software* implementado, foram utilizadas duas JSRs que não fazem parte do conjunto de APIs-padrão distribuídas com o JavaSE: JSR 82 (*Bluetooth*) e JSR 257 (NFC).

Como não fazem parte do conjunto-padrão de APIs, essas implementações podem ser fornecidas por terceiros. No caso da JSR 82, a opção se deu pela BlueCove[1]. Já no caso da JSR 257 não foi possível obter uma implementação que satisfizesse especialmente a necessidade de se poder realizar testes sem o respectivo *hardware*. Nesse caso, optou-se pela realização de uma implementação própria, de modo que atendesse estritamente as necessidades desse projeto.

Apesar de haverem outras implementações que não aderem às especificações JSR, o uso destas se mostrou extremamente vantajoso neste projeto. Para alternar o modo de operação entre *Bluetooth* e emulado, basta alterar em tempo de inicialização qual implementação da JSR 82 deve ser considerada (duas estão disponíveis no *classpath*). Isso faz com que um mesmo código seja executado em implementações distintas da JSR 82.

5.3.1 JSR 82: *Java APIs for Bluetooth*

A escolha da BlueCove se deu pelo fato da mesma aderir à JSR 82, além de possuir implementações para OBEX e suporte ao uso de múltiplos adaptadores *Bluetooth*.

Este último, no entanto, não é um recurso previsto na API da JSR 82. Para evitar que o código do projeto ficasse atrelado à BlueCove, foram criados *wrappers* (adaptadores) que escondem esse recurso específico.

A BlueCove interage diretamente com a biblioteca BlueZ no Linux, via JNI (*Java Native Interface* – Interface Nativa Java). A versão em uso no projeto é a 2.1.0.

[1] http://www.bluecove.org

Adaptadores *Bluetooth* suportados

A BlueCove reconhece todos os adaptadores *Bluetooth* que forem reconhecidos pela BlueZ.

Desta maneira, todos os adaptadores que estiverem disponíveis para a BlueZ estarão disponíveis para o software implementado.

O utilitário `hcitool` da BlueZ permite identificar quais adaptadores foram reconhecidos pelo sistema:

```
user@desktop:~# hcitool dev

Devices:
        hci0    00:03:0D:00:9C:01
        hci1    00:03:0D:00:9C:02
        hci2    00:03:0D:00:9C:03
        hci3    00:03:0D:00:9C:04
        hci4    00:03:0D:00:9C:05
        hci5    00:03:0D:00:9C:06
        hci6    00:03:0D:00:9C:07
        hci7    00:03:0D:00:9C:08
```

Figura 5.1: Listagem de adaptadores *Bluetooth*

No exemplo da figura 5.1 foram encontrados oito adaptadores *Bluetooth*.

Patch OBEX

No decorrer do desenvolvimento do projeto foi detectado um *bug* na BlueCove, no que se refere à inserção de disponibilidade de serviço OBEX no SDDB (*Service Discovery Database* – Base de dados para Descoberta de Serviços).

Esse *bug* afeta implementações que necessitem oferecer serviço OBEX no modo servidor. No caso do projeto, os dispositivos emulados fazem uso desse recurso. Esse *bug* impede que a descoberta de serviços retorne o *service record* (registro de serviço) correspondente ao OBEX ao se realizar uma busca por serviços no dispositivo emulado.

Para que se atingisse o comportamento esperado, foi realizado um *patch* na BlueCove, na classe `ServiceRecordImpl`. O *patch* força a inserção do UUID 0x1105 no SDDB, que corresponde ao OBEX, quando for detectada a disponibilização desse serviço. Esse *patch* não resolve problemas relacionados com outros UUIDs. O JAR da BlueCove que segue com o projeto já possui essa alteração, a reutilização em outros projetos não é recomendada pelo fato do *patch* não ser oficial.

O problema foi relatado sob a identificação 115 para os desenvolvedores da BlueCove.

5.3.2 JSR 257: *Contactless Communication API*

Ao contrário da JSR 82, não foi possível obter uma boa implementação da JSR 257 para JavaSE.

A única implementação *open source* disponível atualmente, a OpenNFC[2], é destinada ao sistema operacional Android, suportando apenas o *hardware* NFC conectado a dispositivos que utilizem esse sistema. Além disso, ainda não foi disponibilizada uma versão da OpenNFC que implemente a JSR 257.

Implementação RMI

A fim de obter uma implementação minimamente funcional da JSR 257, optou-se pelo desenvolvimento de um subprojeto que realiza esse objetivo por meio de Java RMI: *Remote Method Invocation* (Invocação Remota de Métodos).

A implementação simula o uso de um leitor NFC ao disponibilizar um servidor RMI que aceita chamadas `NFCTouchNotify.notifyTouch()`.

Dispositivos *Bluetooth* emulados podem simular um toque no leitor NFC ao chamar esse método, fazendo com que um registro NDEF seja transferido instantaneamente. O *software* *Bluetooth* é notificado imediatamente de cada toque realizado, e com isso pode extrair os parâmetros de conexão com o dispositivo emulado também instantaneamente.

A figura 5.2 mostra que partes da JSR 257 foram implementadas no subprojeto RMI, As partes não-implementadas (por não serem úteis para a aplicação atual) referem-se ao uso de *tags* RFID, cartões ISO 14443 e ao disparamento automático de aplicações JavaME assim que uma ação NFC é detectada para essa aplicação.

É importante notar que o fato dessa implementação RMI fazer uso da JSR 257 faz com que o *software* tenha comportamento idêntico caso fosse feito uso de um leitor NFC real (que também utilize a JSR 257). Isso soma-se ao fato de a JSR 257 ter uma especificação relativamente simples, onde fatores como tempo de transferência de dados, interrupção de leitura e outros não são expostos.

5.4 Abordagem de problemas

Os problemas listados na seção 4.1 são abordados pelo *software* implementado de acordo com a tabela 5.2.

5.5 Fluxos do sistema

As três tarefas principais do *software* são: descoberta de dispositivos, busca de serviços e envio de conteúdos, os quais serão apresentados resumidamente nos diagramas a seguir.

A busca por dispositivos é um processo assíncrono, onde o `DeviceDiscoverer` pede à implementação *Bluetooth* que realize uma busca por dispositivos em uma *thread* separada. À medida em que os dispositivos são encontrados, o `DeviceDiscoverer` é notificado e atualiza um mapa de estados dos dispositivos da vizinhança.

Esse processo repete-se periodicamente, e a duração de cada um é de aproximadamente 10 s, conforme 2.2.2.

[2] http://www.open-nfc.org

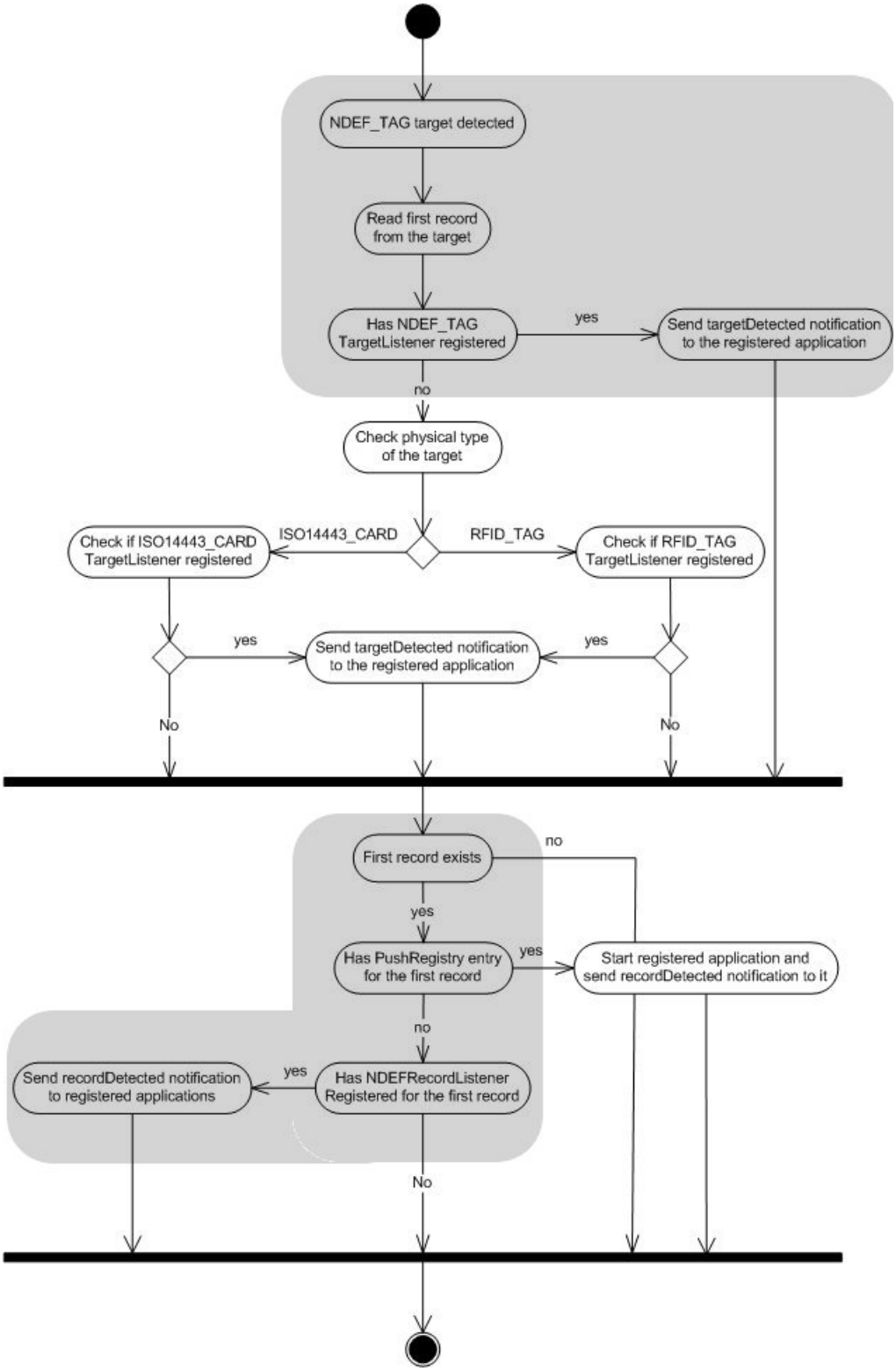


Figura 5.2: Especificação de fluxo de `DiscoveryManager`: apenas a parte destacada foi implementada no subprojeto RMI

Alcance	Depende exclusivamente do adaptador *Bluetooth* instalado no sistema
Não-determinismo da descoberta	Realização periódica e contínua de processos de descoberta, permitindo ao sistema manter um mapa do cenário atual
Cenário de movimentação	São privilegiados os usuários com tecnologia NFC, ficando os demais a cargo do escalonamento de recursos
Limitação da topologia	Suporte a múltiplos adaptadores *Bluetooth*, com número máximo de dispositivos a serem servidos parametrizável
Escalonamento de recursos	São privilegiados os usuários mais detectados em processos de descoberta, com menos erros em buscas de serviço e transmissão de conteúdos e menor recepção de conteúdos

Tabela 5.2: Solução dos problemas da seção 4.1

O processo de busca de serviços também é um processo assíncrono, deflagrado pelo observador `ServiceSearchDispatcher` do processo anterior. Para cada dispositivo descoberto, verifica-se a necessidade de realizar busca de serviços no mesmo (uma vez realizada com sucesso, não há necessidade de repeti-la) junto ao `AdvertisingDeviceManager`, que mantém um mapa atualizado dos dispositivos da vizinhança.

Havendo necessidade de realizar a busca, uma tarefa correspondente é criada (`Service Discoverer`) e enviada ao escalonador de adaptadores *Bluetooth*, que a submeterá ao adaptador adequado ou enfileirará a tarefa para execução posterior. Após a execução de `Service Discoverer`, o mesmo devolve o resultado da busca para `AdvertisingDeviceManager`.

Uma vez com os serviços disponíveis descobertos, o dispositivo passa a estar apto a receber conteúdos multimídia. Nesse caso, o `RunnableAdvertiser` requisita ao `AdvertisingDevice Manager` os dispositivos que são descobertos frequentemente na vizinhança. Dentre esses dispositivos, uma cadeia de filtros (`AdvertisingDeviceFilterChain`) é aplicada de forma a determinar se o dispositivo deve receber conteúdos os não. O objetivo nesse caso é determinar se um dispositivo já recebeu um limite de conteúdos, ou se não deve ser anunciado em qualquer hipótese, por exemplo.

Filtrados os dispositivos de interesse, para cada um deles `FilePushers` são criados para enviar conteúdos, que são alocados nos adaptadores *Bluetooth* por `BluetoothAdapterScheduler`.

Já o serviço de detecção de dispositivos NFC simplifica bastante os processos das figuras 5.3 e 5.4.

O `NFCTouchNotify` é uma implementação que simula um leitor NFC físico, que pode ser tocado por um único dispositivo por vez a qualquer instante. O toque do dispositivo faz com que um registro NDEF seja transferido pelo NFC, sendo que esse registro possui informações de descoberta e busca de serviços. Dadas essas informações, o `NFCDeviceListener` verifica se o dispositivo já é conhecido pelo sistema, consultando o `AdvertisingDeviceManager`. Em caso negativo, insere o novo dispositivo no sistema como se tivesse sido detectado por um processo

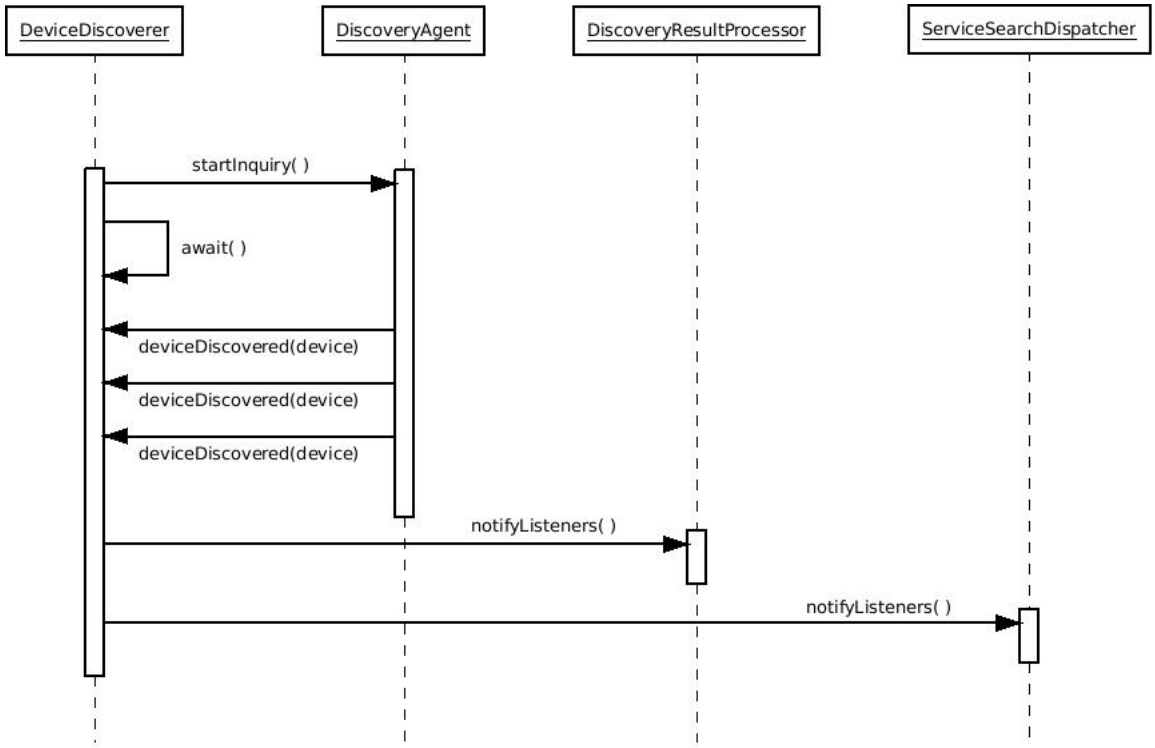


Figura 5.3: Descoberta de dispositivos, sem controle de concorrência e escalonamento

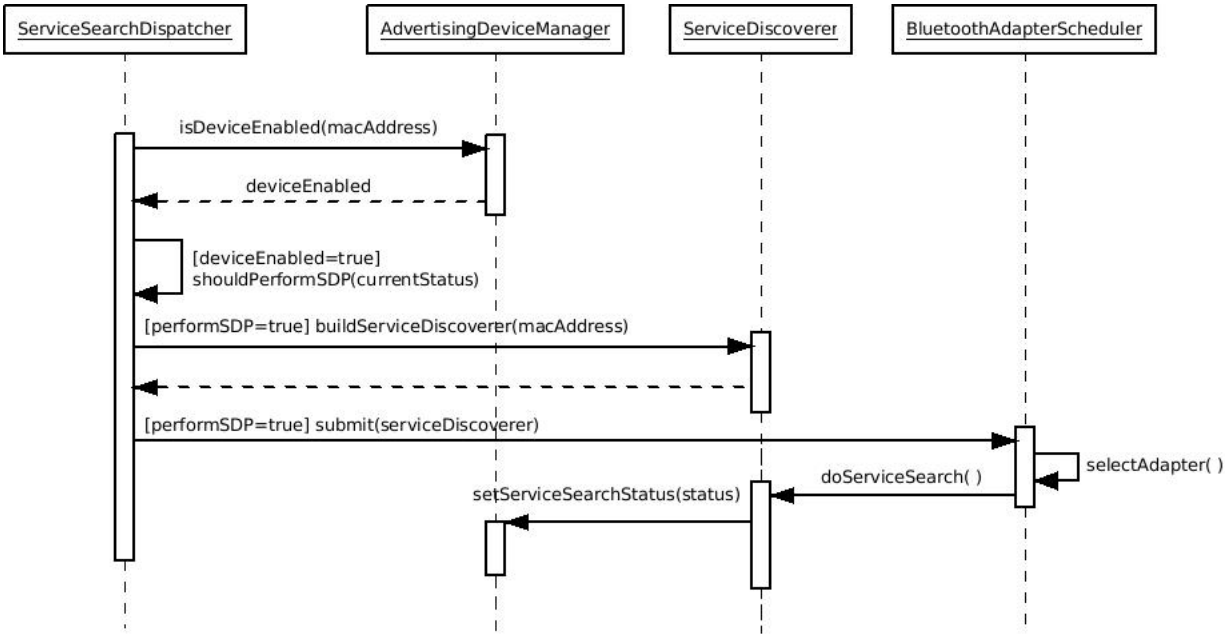


Figura 5.4: Busca de serviços para um único dispositivo

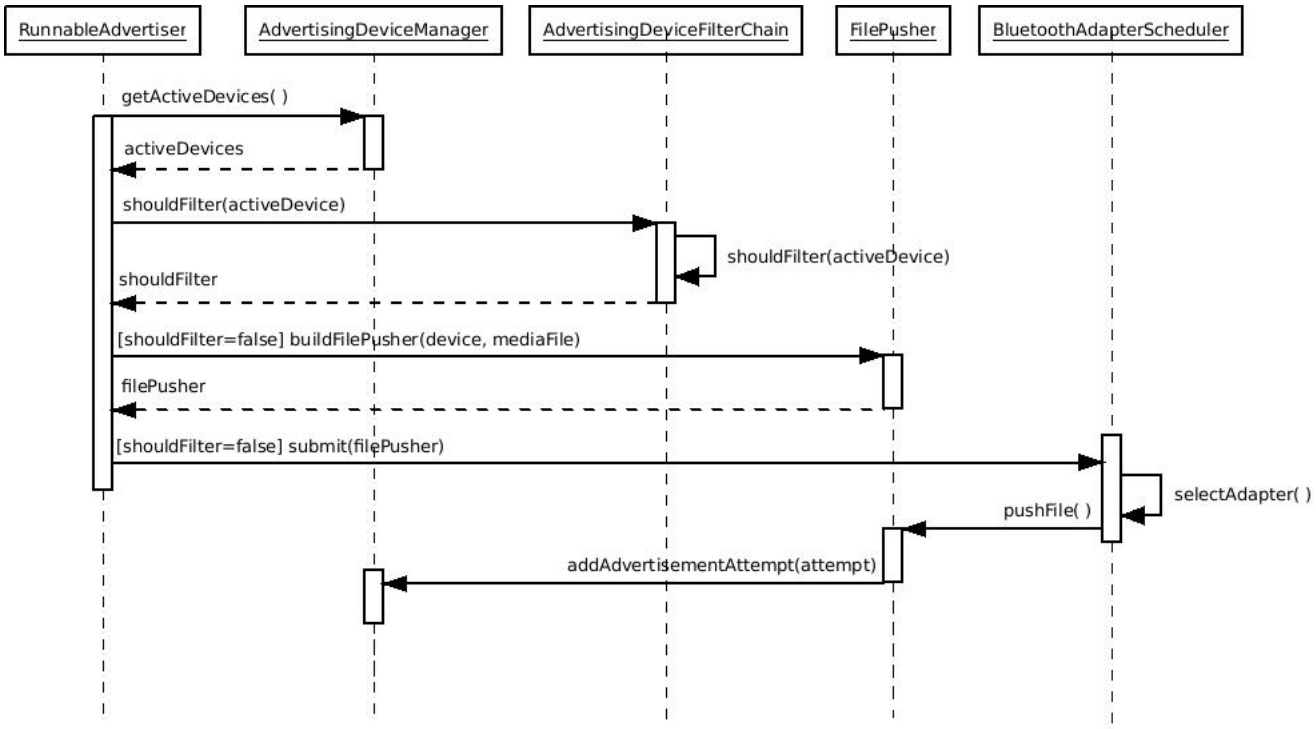


Figura 5.5: Envio de um conteúdo multimídia para um único dispositivo

de descoberta, bem como tivesse sofrido uma busca de serviços logo em seguida. Com isso, o dispositivo passa a estar apto a receber conteúdos multimídia logo em seguida.

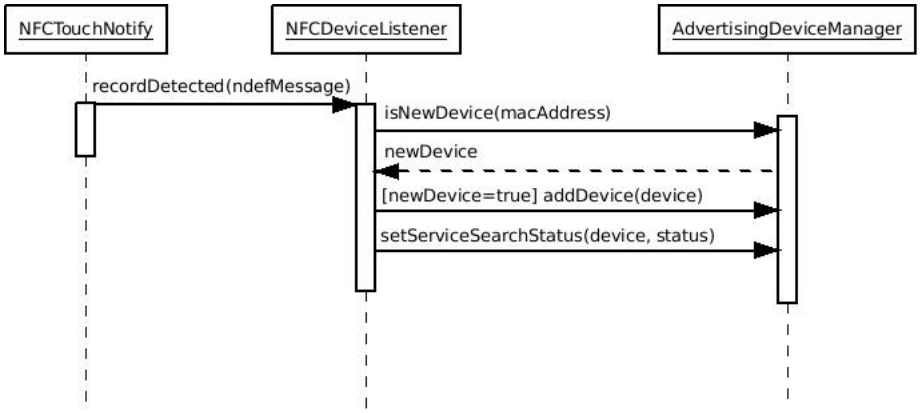


Figura 5.6: Detecção e busca de serviços de dispositivo NFC

5.6 Controle de concorrência e escalonamento

O controle de concorrência é um dos pontos cruciais da implementação. Sem ele a aplicação poderia se tornar inviável com a saturação de uso dos dispositivos *Bluetooth* ou da ocorrência simultânea de determinados processos no mesmo adaptador.

Na implementação o monitor (figura 5.7) deve garantir as seguintes condições:

- O processo de descoberta deve ocorrer de maneira exclusiva em um adaptador;
- Cada adaptador deve suportar no máximo `MAX_CONCURRENT_TRANSFERS` processos de transferência ou busca de serviços simultâneos;
- Transferências de arquivo e buscas de serviço podem ocorrer simultaneamente em um adaptador desde que a soma da ocorrência de ambos não exceda `MAX_CONCURRENT_TRANSFERS`.

O parâmetro `MAX_CONCURRENT_TRANSFERS` é um inteiro arbitrário, que pode ser ajustado de acordo com o desempenho desejado do sistema, porém deve estar entre 1 e 7, que é o máximo que cada adaptador *Bluetooth* suporta [6]. Como a largura de banda de cada adaptador é dividida entre os aparelhos receptores, quanto menor o número, maior a velocidade de entrega de conteúdos. Porém, se houver necessidade da entrega de muitos conteúdos, a fila de espera pode ser grande se `MAX_CONCURRENT_TRANSFERS` for muito reduzido.

A escolha dos adaptadores de modo a lidar com a saturação dos dispositivos e a ocorrência de processos de descoberta deve ficar a cargo de um escalonador externo. Na implementação foi utilizado um simples escalonador *round-robin*, que faz a escolha sequencial e cíclica dos adaptadores a serem utilizados.

5.7 Emulação e ambiente de simulação

A fim de realizar testes no sistema em situações mais próximas do real, com centenas de dispositivos, foi desenvolvido o subprojeto **Bluetooth/NFC Emulator**.

Esse subprojeto utiliza a biblioteca BlueCoveEmu[3], que provê uma camada de emulação da JSR 82 também sobre RMI.

5.7.1 Dispositivos virtuais

A BlueCoveEmu permite que dispositivos virtuais sejam criados por meio de uma API própria. No entanto, a programação de cada dispositivo é feita normalmente via JSR 82.

Os dispositivos são implementados de maneira a simular as seguintes características:

- **Descoberta:** visível, invisível, descoberta limitada, aleatoriamente visível;
- **Aceitação de conteúdo:** sempre aceita, sempre rejeita;
- **Velocidade:** pode ser especificada de maneira a simular a distância ao transmissor;
- **Transferência:** contínua ou com erros;
- **Busca de serviço:** possui ou não o registro para OBEX *push*;
- **Capacidades:** com ou sem NFC.

[3] http://bluecove.org/bluecove-emu/

```
monitor BluetoothAdapterMonitor {
     int  discoveryPending = 0;
     cond discoveryCondition;
     cond transferCondition;
     cond semaphoreConditions[1:ADAPTERS-1];
     sem  discoverySemaphore = MAX_CONCURRENT_TRANSFERS;
     sem  adapterSemaphores[1:ADAPTERS-1] =
                    ([ADAPTERS-1] MAX_CONCURRENT_TRANSFERS);

     procedure acquireAdapterForDiscovery() {
         discoveryPending = 1;
         while (discoverySemaphore != MAX_CONCURRENT_TRANSFERS)
             wait(discoveryCondition);
         discoverySemaphore = 1;
         P(discoverySemaphore);
     }

     procedure releaseAdapterForDiscovery() {
         V(discoverySemaphore);
         discoverySemaphore = MAX_CONCURRENT_TRANSFERS;
         discoveryPending = 0;
         signal(transferCondition);
     }

     procedure acquireAdapter(int adapterNumber) {
         if (adapterNumber == DISCOVERY_ADAPTER) {
             while (discoveryPending == 1)
                 wait(transferCondition);
         }
         while (adapterSemaphores[adapterNumber] == 0)
             wait(semaphoreConditions[adapterNumber]);
         P(adapterSemaphores[adapterNumber]);
     }

     procedure releaseAdapter(int adapterNumber) {
         V(adapterSemaphores[adapterNumber]);
         signal(semaphoreConditions[adapterNumber]);
         signal(discoveryCondition);
     }
}
```

Figura 5.7: Esquema do monitor para controle de acesso aos adaptadores *Bluetooth*, de acordo com notação de [1]

Uma instância de um dispositivo virtual pode combinar essas características, desde que sejam interessantes para a simulação.

Situações/características	Descoberta	Aceitação	Velocidade	Transferência	Serviços	NFC
Erro de busca de serviços disponíveis	visível	aceita	alta	com erros	sem registro	sim/não
Distância ao transmissor	aleatória	aceita	baixa	com erros	com registro	sim/não
Dispositivo NFC não-encontrado	invisível	aceita	alta	contínua	com registro	sim
Rejeição de conteúdos	visível	rejeita	alta	contínua	com registro	não
Usuário passageiro	visível	aceita	baixa	com erros	com registro	não

Tabela 5.3: Algumas situações possíveis de serem simuladas com as características implementadas nos dispositivos virtuais

A tabela 5.3 mostra como algumas situações podem ser simuladas utilizando estas características.

5.7.2 Deficiências do ambiente de simulação

Apesar de ser bastante adequada para a realização de testes unitários de código JSR 82, a BlueCoveEmu não permite simular um grande número de condições adversas.

No caso, seria interessante poder interromper o fluxo de transferência de arquivo, de maneira a simular uma saída da área de atuação *Bluetooth*. No entanto, na ocasião de transferência de dados a BlueCoveEmu realiza um `System.arrayCopy()` entre o *software Bluetooth* e o dispositivo virtual (ver `EmulatorRFCommClient.write():65`). Como os dados não são transferidos em blocos, não é possível realizar a interrupção do *stream* no meio da transferência.

Outra situação que não é simulada pela biblioteca é quanto ao alcance dos dispositivos *Bluetooth*. Uma vez criados os dispositivos virtuais, os mesmos estarão sempre ao alcance.

Capítulo 6

Simulações

Para este projeto, especificamente, é inviável verificar o comportamento do sistema de *Bluetooth marketing* com dispositivos reais.

Seriam necessários centenas de dispositivos habilitados com *Bluetooth*, e cada um deveria receber um *software* que lhe desse um comportamento automatizado (aceitação/rejeição automática de conteúdos, toque em leitor NFC etc).

Por essa razão foi desenvolvido o ambiente de simulação, de maneira que possam ser simulados um número qualquer de dispositivos em diversas condições do ambiente.

6.1 Objetivo

O objetivo das simulações é observar a influência do NFC sobre o comportamento do sistema de *Bluetooth marketing*, inicialmente sobre perturbações isoladas dos cenários de simulação, em seguida sobre cenários que sejam mais próximos do observado na realidade.

6.2 Metodologia

As simulações foram realizadas utilizando o modo de emulação do *software* desenvolvido.

Os passos envolvidos na execução das mesmas compreende:

1. Definição do cenário de simulação;
2. Parametrização e geração de arquivos de cenários de simulação;
3. Execução do *software* de *Bluetooth marketing* no modo emulado sobre o cenário definido;
4. Coleta de resultados na base de dados SQL.

Para cada cenário, a execução da simulação é feita uma única vez.

6.2.1 Definição do cenário de simulação

Um cenário de simulação envolve o sistema de transmissão de conteúdos *Bluetooth marketing*, um ou mais leitores NFC e os dispositivos simulados.

A definição geral do cenário se dá pela escolha do número de participantes da simulação e pela distribuição porcentual de características entre eles. Fora isso, planejam-se o número de adaptadores *Bluetooth* utilizados e o número de leitores NFC empregados.

6.2.2 Parametrização e geração de arquivos de cenários de simulação

Feita essa definição geral, é necessário gerar os arquivos com os cenários de simulação, bem como realizar a parametrização do transmissor de conteúdos e leitores NFC.

Neste projeto, os arquivos de simulação são gerados a partir da definição geral pela classe `SimulationScenarioFileGenerator`.

Os parâmetros de cada um dos componentes da simulação podem ser combinados de acordo com as definições a seguir.

Parâmetros do transmissor de conteúdos

A seção B.2.1 detalha todas as opções de configuração do arquivo `bluetooth.properties`.

No modo de emulação é possível especificar quantos adaptadores *Bluetooth* serão utilizados para a simulação, por meio da *flag* `-adapters`, conforme seção B.3.

Parâmetros e comportamentos dos dispositivos receptores

A figura 6.1 ilustra o formato geral de um arquivo de simulação, onde é possível especificar os dispositivos emulados, de `0` a `N`.

- **Endereço MAC** (`device.N`)
  Identificador único no formato `001122334455`.
- **Modo de descoberta** (`device.N.discoverable_mode`)
  Define o modo de visibilidade do dispositivo: visível (`visible`), oculto (`hidden`) ou aleatório (`random`). Optando pelo modo aleatório, deve-se especificar o intervalo de descoberta aleatória.
- **Visível após** (`device.N.discoverable_after`)
  Após quanto tempo, em segundos, o dispositivo torna-se visível na área de atuação *Bluetooth*, em segundos. A principal utilidade é simular chegadas de dispositivos em tempos distintos.
- **Descoberta aleatória** (`device.N.random_discovery`)
  Especificação de um intervalo, em segundos, no qual o dispositivo alterna entre modo visível e oculto. A duração do intervalo é sorteada de maneira uniforme de 0 a `random_discovery`.

```
device.0=000000000011
device.0.discoverable_mode=visible
device.0.discoverable_after=0
device.0.random_discovery=0
device.0.transfer_rate=2500000
device.0.nfc_capable=false
device.0.nfc_touch_after=10
device.0.incoming_transfer=accept
device.0.sdp_enabled=true

...

device.N=0000000000XX

...
```

Figura 6.1: Formato geral do arquivo de cenário de simulação

- **Taxa de transferência** (`device.N.transfer_rate`)
  Taxa de transferência, em bps, que cada dispositivo deve receber. Normalmente um dispositivo não realiza transferências em sua capacidade máxima, devido a interferências e atenuações comuns em comunicações sem fio.

- **Habilitado com NFC** (`device.N.nfc_capable`)
  Especifica se o dispositivo tem capacidade NFC ou não (`true` ou `false`). Tendo a capacidade, é opcional especificar o parâmetro de intervalo até toque NFC. Por padrão, é 0.

- **Intervalo até toque NFC** (`device.N.nfc_touch_after`)
  Tempo de espera até que um dispositivo habilitado com NFC decida realizar o toque em um leitor. Permite que toques ocorram em diferentes instantes.

- **Aceitação de conteúdos** (`device.N.incoming_transfer`)
  Comportamentos diante de uma solicitação de transferência de arquivos: aceitar (`accept`), rejeitar (`reject`) ou corromper (`corrupt`). A corrupção de transferência simula a saída da área de atuação, interferência que impeça a comunicação ou cancelamento voluntário do processo. Ocorre em um tempo aleatório sorteado uniformemente no intervalo de tempo mínimo de uma transferência.

- **Habilitação de SDP** (`device.N.sdp_enabled`)
  Habilita (`true`) ou desabilita (`false`) a busca de serviços em cada dispositivo, para simular eventuais falhas em busca de serviço.

Parâmetros dos leitores NFC emulados

A seção B.2.3 descreve como configurar os leitores NFC emulados.

6.2.3 Execução do *software* de *Bluetooth marketing* no modo emulado

A execução do *software* em modo emulado é praticamente idêntica à execução do sistema com adaptadores *Bluetooth* reais. Basta deixar o sistema executando que ele encontrará os dispositivos emulados e procederá com o envio de conteúdos aos mesmos.

Cada tentativa de envio de conteúdo, bem como seus resultados (sucesso, falha, rejeição etc) são registrados em uma base de dados SQL para consulta posterior.

O sistema é finalizado quando todas as tentativas de veiculação de conteúdos cessarem (aceitação total e rejeição total pelos diversos motivos).

6.2.4 Coleta de resultados na base de dados SQL

Os resultados sobre cada execução das simulações podem ser obtidos por meio de consultas quaisquer à base de dados do *software*.

A tabela de provavelmente maior interesse é a `ADVERTISINGMETADATA` (figura 6.2), que registra cada tentativa de veiculação de conteúdos.

```
CREATE TABLE ADVERTISINGMETADATA (
     ID          NUMERIC(19)   NOT NULL,
     MACADDRESS  VARCHAR(12)   NOT NULL,
     FILENAME    VARCHAR(255)  NOT NULL,
     PUSHSTATUS  VARCHAR(32)   NOT NULL,
     STARTTIME   TIMESTAMP     NOT NULL,
     ENDTIME     TIMESTAMP     NOT NULL,
     PRIMARY     KEY (ID)
)
```

Figura 6.2: Definição da tabela `ADVERTISINGMETADATA`

Outras tabelas disponíveis são `DISCOVERYMETADATA`, que registra o resultado de cada processo de descoberta, e `SERVICESEARCHMETADATA`, com o resultado da busca de serviços em cada dispositivo.

6.3 Problemas e soluções de contorno

Apenas durante as primeiras execuções das simulações é que foi detectado um *bug* na biblioteca BlueCoveEmu, o qual inviabiliza o uso de múltiplas transferências em múltiplos adaptadores *Bluetooth* virtuais.

Para viabilizar as simulações foi adotada uma solução de contorno, que consiste em limitar uma única transferência por adaptador *Bluetooth* virtual. Para simular n múltiplas transferências por adaptador, o número de adaptadores é multiplicado por n.

Por exemplo, caso se deseje simular quatro adaptadores *Bluetooth* com no máximo quatro transferências simultâneas por adaptador, são utilizados 16 adaptadores virtuais com no máximo uma transferência por adaptador.

A fim de que cada dispositivo emulado não use o máximo de banda por adaptador, o mesmo tem sua largura de banda dividida por n. Desse modo, um dispositivo capaz de transferir a 2 Mbps tem sua taxa de transferência máxima definida em 500 kbps, caso fosse feita a simulação com no máximo quatro transferências por adaptador.

6.4 Limitações

É praticamente impossível simular com verossimilhança um ambiente tão dinâmico e pouco determinístico envolvendo comunicações sem fio.

A própria biblioteca BlueCoveEmu apresenta algumas limitações nesse sentido, como a taxa de transferência, que é praticamente instantânea. Outros exemplos são a ausência de erros, problemas de transferência, entre outros.

Várias dessas omissões da biblioteca foram implementadas de maneira a se aproximar de um cenário real. No entanto, pela dificuldade em reproduzi-lo com exatidão, as simulações executadas buscarão mostrar cenários de melhor e pior caso, que são mais facilmente reproduzíveis na prática.

6.5 Simulação 1: efeito apenas da rejeição ativa

Para estas simulações foi escolhida a observação da **espera até o início da transferência** (média, máxima e mínima), que indica quanto tempo se passa até que o dispositivo-alvo receba o pedido de autorização para início de transferência de dados.

Normalmente este é um parâmetro importante em relação à experiência do usuário. Uma vez que o mesmo adentre a área de atuação *Bluetooth*, ele espera receber o conteúdo o quanto antes.

6.5.1 Parâmetros fixos

A tabela 6.1 mostra os parâmetros fixados para estas simulações:

Optou-se pelo **tempo de chegada simultâneo** para simular uma situação crítica, que é quando o sistema de *Bluetooth marketing* é ativado em meio a muitos dispositivos-alvo. Neste caso, ele deve trabalhar de forma a servir mais rapidamente aqueles que tem mais propensão a receber os conteúdos multimídia.

Fixar o **número de adaptadores** visa sobrecarregá-los conforme o número de dispositivos-alvo aumenta. Com isso, o sistema poderá perder mais tempo tentando enviar para dispositivos que rejeitam o conteúdo sumariamente. Com o uso do NFC, o esperado é que aqueles que fizerem o toque no leitor sejam priorizados em meio aos que certamente negarão o recebimento.

Tempo de chegada	simultâneo
Número de adaptadores	4
Máx. transferências por adaptador	4
Leitores NFC	1
Intervalo entre toques NFC	5 s
Visibilidade de dispositivos com NFC	ocultos
Rejeição ativa	25%

Tabela 6.1: Parâmetros fixos para simulação de 6.5

Já a fixação de um **único leitor NFC** busca visualizar de que maneira a dependência do toque NFC pode influenciar o sistema. Uma vez que todos os dispositivos não podem realizar o toque simultaneamente, uma eventual espera em fila para realizar o toque pode ocorrer. Na simulação os toques foram fixados em intervalos uniformes de 5 s.

Uma restrição importante é que nesta simulação os dispositivos habilitados com NFC estão **ocultos** para o processo de descoberta *Bluetooth*. Portanto, a recepção de conteúdos depende exclusivamente do toque NFC.

A **rejeição ativa** ocorre quando um usuário recebe a requisição de transferência *Bluetooth* mas opta por não efetuá-la. Outras situações de rejeição podem ocorrer quando o usuário não responde à permissão para transferir conteúdo (nesse caso ocorre rejeição por tempo de espera – *timeout*) ou recebe a requisição mas deixa de respondê-la dentro da área de atuação *Bluetooth*, por exemplo.

Para a simulação, a rejeição ativa ocorre num intervalo de 0 a 30 s, sorteado uniformemente.

6.5.2 Tempos de espera sem e com NFC

Os resultados das simulações encontram-se nas tabelas 6.2 e 6.3.

Número de dispositivos	**10**	**25**	**50**	**100**
Tempo médio (s)	52	44	43	47
Tempo mínimo (s)	50	42	23	19
Tempo máximo (s)	60	56	111	93

Tabela 6.2: Tempo médio de espera sem NFC apenas com rejeição ativa

Número de dispositivos	**10**	**25**	**50**	**100**
Tempo médio (s)	23	40	36	63
Tempo mínimo (s)	15	31	14	26
Tempo máximo (s)	33	51	93	143

Tabela 6.3: Tempo médio de espera com NFC e rejeição ativa

O gráfico da figura 6.3 mostra que em relação ao tempo médio de espera, o uso do NFC apresenta redução significativa até o limite de 50 dispositivos.

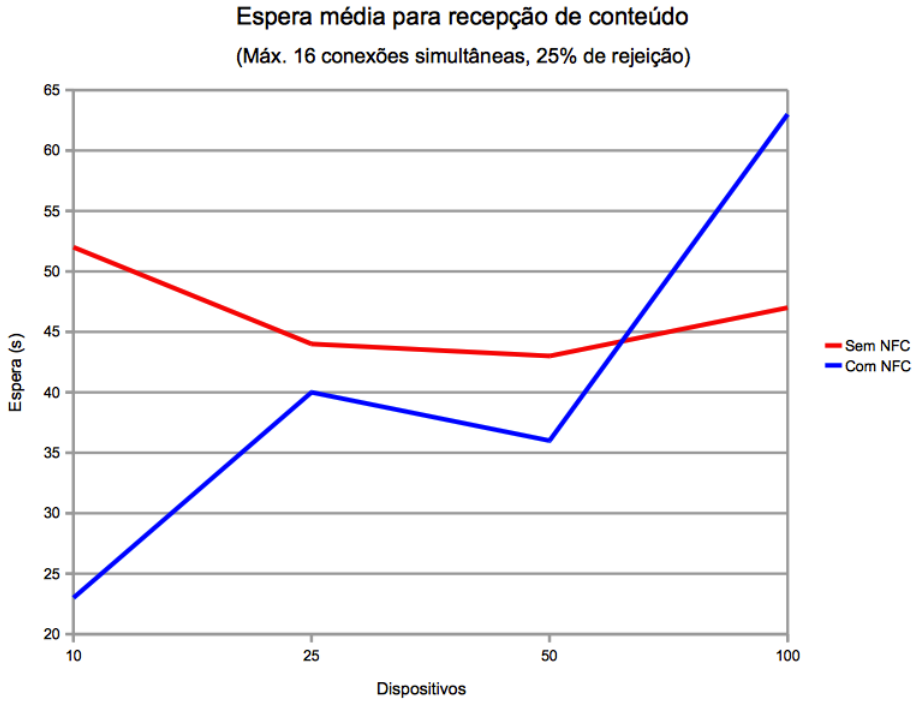


Figura 6.3: Tempo média de espera (em s), apenas com a interferência da rejeição ativa

Conforme a tabela 6.1, o número de adaptadores *Bluetooth* é fixo para qualquer número de dispositivos. Isso significa que quanto mais dispositivos devem ser servidos, a taxa de ocupação desses adaptadores tende a ser maior.

Também segundo a tabela 6.1 o número de transferências simultâneas é limitado. Sejam n o número de adaptadores *Bluetooth*, s o número máximo de transferências simultâneas e b o número de dispositivos *Bluetooth* prontos para serem servidos em um determinado instante. Caso $n \times s < b$ alguns dispositivos devem aguardar a liberação de um transmissor para que possam receber o conteúdo multimídia. Quanto maior b, mais dispositivos ficam nessa espera, aumentando portanto o tempo médio de espera.

A figura 6.3 ilustra exatamente essa situação. Enquanto o sistema possui capacidade de atender todos os dispositivos *Bluetooth*, o uso do NFC mostra-se vantajoso na redução do tempo de espera. Porém, quando o número de dispositivos a ser servido é muito maior que o suportado pela configuração, o uso do NFC não é vantajoso, por sobrecarregar ainda mais o sistema.

6.6 Simulação 2: aproximação de cenário real

O objetivo da Simulação 1 era observar o efeito do uso do NFC em um cenário quase ideal, com apenas um fator de interferência, no caso a rejeição ativa.

Já nesta simulação pretende-se simular um cenário mais próximo do real, onde a quantidade de interferências é bastante significativa. Para isso fixou-se uma proporção onde 75% dos dispositivos presentes no cenário de simulação impõem alguma perturbação no sistema.

Novamente será feita a observação da **espera até o início da transferência**.

6.6.1 Parâmetros

Em relação à Simulação 1, quatro parâmetros apresentam variação: número de adaptadores e de leitores NFC, visibilidade de dispositivos, bem como a redução do intervalo entre toques NFC.

Já as interferências dividem-se em três grupos, cada um contribuindo com 25% do total de dispositivos: rejeição ativa, erro de transmissão e visibilidade aleatória.

Simulações	**A**	**B**	**C**	**D**	**E**	**F**
Tempo de chegada	simultâneo					
Número de adaptadores	4		16	4	16	64
Máx. transferências por adaptador	4					
Leitores NFC	4					
Intervalo entre toques NFC	2 s					
Visibilidade de dispositivos com NFC	ocultos			visíveis		
Dispositivos com NFC	0%	25%				
Rejeição ativa	25%					
Erro de transmissão	25%					
Visibilidade aleatória	25%					

Tabela 6.4: Parâmetros de simulações de 6.6

O **número de adaptadores** é um aspecto importante em qualquer sistema de *marketing* de proximidade, já que tanto a largura de banda quanto o número de dispositivos que podem ser servidos simultaneamente dependem deste fator. Caso o sistema tenha bastante sucesso em encontrar dispositivos-alvo capazes de receber conteúdos, o gargalo pode estar no número de transmissores.

Já o **número de leitores NFC** influencia o número de dispositivos-alvo encontrados e aptos a receber com sucesso a transmissão de conteúdos. Com a eliminação da restrição de um único ponto de entrada NFC podem ocorrer toques simultâneos, aumentando assim a taxa de descoberta desse tipo de dispositivos.

A **redução do intervalo entre toques NFC** colabora diretamente com o parâmetro anterior. Uma vez que aumenta o número de leitores NFC, o número de toques pode ter sua frequência aumentada.

Para observar os efeitos da dependência exclusiva do toque NFC para dispositivos habilitados com essa tecnologia é que nesta simulação foi testada a ativação da **visibilidade** dos mesmos. Com isso, um dispositivo passa a estar apto a receber conteúdos tanto pelo toque NFC quanto pelos processos de descoberta e busca de serviços, o que vier primeiro.

Dentre os fatores de interferência, o **erro de transmissão** faz com que dispositivos aptos a receber transferência de conteúdos deem início à mesma porém não a completam. Desta maneira, dispositivos que tenham apresentado esse tipo de comportamento uma ou mais vezes devem ter menos prioridade que aqueles capazes de receber os conteúdos integralmente.

Por fim, a **visibilidade aleatória** simula um fator de indeterminismo no sistema. Mesmo que encontrado um dispositivo, não é garantido que o mesmo estará sempre na área de atuação *Bluetooth* no momento em que o sistema alocar uma transmissão para o mesmo. Portanto, dispositivos com maior visibilidade devem ter prioridade sobre estes, já que tem mais chances de efetuar a transmissão com sucesso.

6.6.2 Tipos de simulação e resultados

Foram realizadas seis simulações, cujos resultados encontram-se na tabela 6.5.

Dispositivos	10	25	50	100	150	200
Simulação A	48	34	38	60	106	150
Simulação B	22	22	38	75	155	185
Simulação C	16	20	22	34	48	65
Simulação D	13	28	36	74	120	154
Simulação E	3	20	23	24	43	37
Simulação F	29	20	21	25	22	22

Tabela 6.5: Tempo médio de espera (em s) com NFC e cenário com 75% dos dispositivos em condição desfavorável

Dispositivos	10	25	50	100	150	200
Simulação A	4,763	13,925	21,383	47,180	87,093	123,057
Simulação B	18,345	13,462	16,717	43,271	80,920	100,942
Simulação C	3,686	7,324	7,430	27,281	28,924	38,647
Simulação D	3,686	20,657	18,837	52,915	92,660	125,019
Simulação E	2,165	7,582	12,555	9,894	28,984	20,953
Simulação F	8,133	6,115	5,841	8,540	6,400	9,442

Tabela 6.6: Desvio-padrão do tempo de espera (em s) com NFC e cenário com 75% dos dispositivos em condição desfavorável

Dispositivos	10	25	50	100	150	200
Simulação A	46	23	21	20	20	20
Simulação B	8	7	16	10	19	21
Simulação C	11	12	9	7	9	11
Simulação D	8	11	10	11	9	11
Simulação E	0	11	11	9	8	7
Simulação F	20	11	11	13	11	9

Tabela 6.7: Tempo mínimo de espera (em s) com NFC e cenário com 75% dos dispositivos em condição desfavorável

Dispositivos	10	25	50	100	150	200
Simulação A	57	59	97	212	310	401
Simulação B	61	52	74	165	323	433
Simulação C	20	42	40	153	131	169
Simulação D	17	85	82	181	329	475
Simulação E	5	42	81	62	160	100
Simulação F	40	31	30	54	43	56

Tabela 6.8: Tempo máximo de espera (em s) com NFC e cenário com 75% dos dispositivos em condição desfavorável

Para facilitar a visualização e interpretação, os dados da tabela 6.5 foram desmembrados em dois gráficos: um agrupando os dados de dispositivos sem NFC ou com NFC e invisível ao processo de descoberta *Bluetooth* e outro com dispositivos com NFC e ao mesmo tempo visíveis ao processo de descoberta.

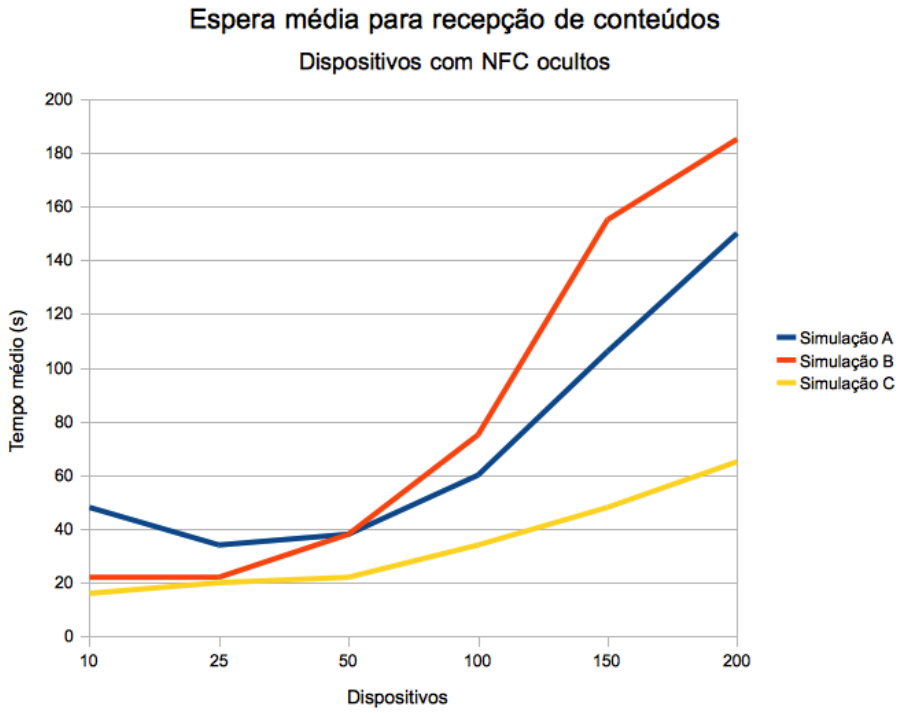


Figura 6.4: Tempo médio de espera (em s) para dispositivos com NFC e invisíveis ao processo de descoberta *Bluetooth*

A Simulação A, conforme a figura 6.4, é a que apresenta o pior resultado em termos de espera média até recepção de conteúdos. Isso se deve unicamente ao fato do número de transmissores ser muito reduzido, já que nessa simulação não há dispositivos com NFC.

Já as simulações B e C mostram uma melhora no tempo de espera com o uso de NFC.

No caso da simulação B, a melhora não é tão acentuada pela limitação do número de transmissores frente ao número de dispositivos a serem servidos. Já quadruplicando esse número de

transmissores (Simulação C), é possível ver uma melhora significativa. Para 200 dispositivos, a diferença na espera média chega a 478%.

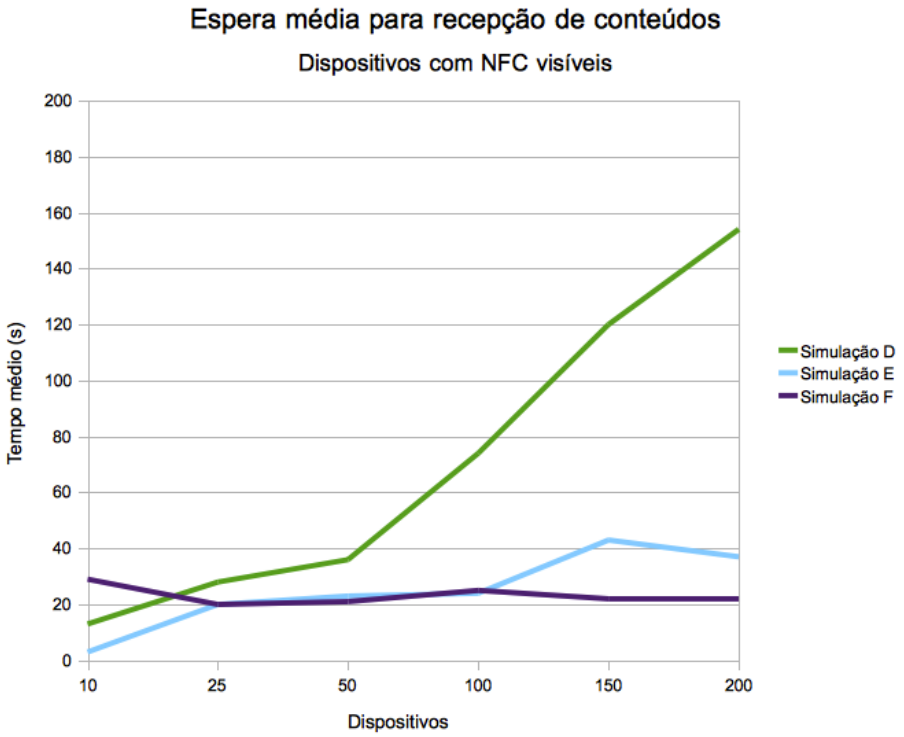


Figura 6.5: Tempo médio de espera (em s) para dispositivos com NFC mas mas ao processo de descoberta *Bluetooth*

O gráfico da figura 6.5 ilustra situações onde os dispositivos com NFC não dependem apenas do toque para serem descobertos: estão também visíveis para o processo de descoberta *Bluetooth*. Nesse caso, os mesmos podem ser descobertos e terem busca de serviço antes mesmo de efetuarem o toque, estando portanto prontos para receberem o conteúdo multimídia.

A Simulação F é apenas ilustrativa, uma vez que usa um número de transmissores fora da realidade. A intenção é mostrar que tendo um número suficiente de leitores NFC e transmissores *Bluetooth*, os dispositivos são servidos de maneira praticamente instantânea. Sendo esse o ideal, o esperado é que as simulações que envolvam aumento do número de transmissores se aproximem da Simulação F.

A Simulação D mostra que mesmo que os dispositivos com NFC estejam visíveis, o número de transmissores continua sendo um limitante, assemelhando-se assim à Simulação A.

Já a Simulação E mostra o que parece ser o cenário ideal, já que obteve praticamente todas as melhoras marcas no tempo médio de espera.

6.7 Conclusões

Os resultados das simulações mostram que ao se considerar adotar NFC para reduzir o tempo de espera em um sistema de *marketing* de proximidade, os seguintes fatores devem ser levados

em conta:

1. Número de transmissores (adaptadores) *Bluetooth*;
2. Número de leitores NFC;
3. Número de dispositivos-alvo.

O fator (3) é praticamente determinante de todos os outros, uma vez que especialmente pela seção 6.6 pode-se perceber que para um número pequeno de dispositivos o uso de NFC é dispensável.

O uso de NFC, portanto, tem melhores resultados em ambientes onde há grande número de dispositivos habilitados com *Bluetooth*. Nestes ambientes, quanto maior for a demanda por recepção de conteúdos, maior deve ser a implantação de transmissores *Bluetooth* e leitores NFC.

Capítulo 7

Parte subjetiva

As seções a seguir buscam mostrar como o trabalho e curso de Ciência da Computação se relacionam.

7.1 Desafios e frustrações

No desenvolvimento do *software Bluetooth* dois problemas que apareceram no decorrer do desenvolvimento ameaçavam o cumprimento do cronograma: um *bug* na biblioteca BlueCove e a ausência de uma implementação JSR 257 (NFC), ao contrário do que se acreditava na época da elaboração do plano de trabalho.

A biblioteca BlueCove possui um conjunto de testes incrivelmente amplo e automatizado, somando-se a isso práticas de integração contínua. Não era de se esperar que uma simples implementação de OBEX *push server* fosse escapar a todos esses testes.

Mas escapou e causou transtornos por dias.

Inicialmente se pensa que o problema é no próprio *software* desenvolvido, e não na biblioteca utilizada. Foram necessários dias para convencimento do contrário e então o início de um processo de depuração da biblioteca para enfim poder encontrar e corrigir o *bug* na mesma, e só então prosseguir com o cronograma.

Felizmente na contramão de todos esses transtornos vem o fato da biblioteca ser *software* livre, o que certamente contribui para um desfecho favorável.

Já a falta de uma implementação JSR 257 quase forçou uma alteração no título do trabalho para: "Bluetooth: estudo de caso".

As implementações disponíveis atualmente ou dependem muito de um *hardware* específico, ou são para sistemas que não são o alvo deste projeto (Android, por exemplo) ou não possuem uma maneira de realizar emulação ou testes automatizados, que são essenciais para simulações de larga escala.

Uma vez que a falta desse recurso tornaria o trabalho um pouco menos interessante, foi tomada a decisão de realizar uma implementação própria da JSR 257, que provesse estritamente os recursos necessários.

Apesar de ser uma especificação relativamente curta (94 páginas de [7], contra as 1420 de [10]), é um pouco intimidador pegar um documento desses e sair implementando do zero. No entanto, com o auxílio de um exemplo de uso da JSR 257 postado na página da Oracle, foi

possível fazer uma implementação *top-down* da especificação.

Todos esses foram desafios, a frustração talvez fique por conta do aspecto "inacabado" da JSR 257 implementada. Seria interessante poder desenvolver um *framework* (arcabouço) que tivesse a mesma utilidade da BlueCoveEmu, para desenvolvedores que trabalham com Java e NFC, dada a falta de ferramentas no mercado que forneçam esse ambiente de emulação e testes automatizados.

7.2 Disciplinas mais relevantes para o trabalho

A seguir, algumas das disciplinas mais relevantes para o desenvolvimento deste trabalho.

MAC0463 – Computação móvel

Apesar de já haver uma inclinação pessoal para tratar de assuntos da área de mobilidade, essa disciplina só consolidou esse interesse.

Na ocasião houve a oportunidade de elaborar uma monografia sobre RFID, que possui um relacionamento muito estreito com o NFC do qual trata esse trabalho.

Fora isso a programação para dispositivos móveis também foi algo que despertou interesse e colaborou no desenvolvimento dos dispositivos virtuais emulados, por exemplo, que usam um programa que pode ser facilmente instalado em um aparelho celular (já que a BlueCove implementa uma parte da API JavaME), agindo desta maneira como um dispositivo real para testes.

MAC0438 – Programação Concorrente

Não é exagero dizer que sem os fundamentos lançados por essa disciplina, o *software* seria inviável.

Conceitos como *race conditions*, monitores, variáveis de condição, semáforos, *deadlocks*, *threadpools*, *test-and-set* e outros foram essenciais tanto na concepção do projeto quanto na execução do mesmo.

Apesar de o controle de concorrência utilizado ser relativamente simples, a falta do mesmo no *software* poderia trazer resultados indesejáveis como: infinidades de *threads* sendo criadas fora de um *pool* e consumindo recursos desnecessários, ocorrência simultânea de eventos incompatíveis em um mesmo adaptador *Bluetooth*, tentativa de superar o limite de conexões simultâneas, entre outros.

Em um teste prático foi possível observar o alto uso de CPU pela criação de *threads* a cada necessidade de uso (cerca de 66% de ocupação), assim como várias exceções lançadas pela biblioteca *Bluetooth* referentes à limites excedidos ou tarefas concorrentes incompatíveis.

MAC0441 – Programação Orientada a Objetos

Esta disciplina essencialmente apresentou padrões de projeto, muitos dos quais foram utilizados na implementação do *software*, tratando-se de soluções comprovadas para problemas recorrentes.

Padrões como (segundo [4]) *Observer*, *Factory Method*, *Adapter*, *Chain of Responsibility*, *Singleton*, *Strategy*, foram reconhecidos e aplicados.

Muitas práticas referentes a programação para interfaces, uso ou não de herança, modelagem de objetos, diagramas UML, também tiveram aplicação imediata neste trabalho.

Outras práticas referem-se a refatoração de código e elaboração de testes.

MAC0440 – Sistemas de Objetos Distribuídos

Conceitos como chamada remota de métodos (RMI) e uso de *stubs* foram importantes aprendizados nesta disciplina, ainda que sejam apenas uma parte reduzida da mesma.

Esses conceitos foram utilizados diretamente na implementação da JSR 257, que utiliza RMI para simular o toque de um dispositivo NFC em um leitor.

MAC0439 – Laboratório de Bancos de Dados

Na disciplina foi dada uma abordagem prática dos conceitos aprendidos em MAC0426 – Sistemas de Bancos de Dados.

Essa abordagem foi utilizada diretamente no desenvolvimento da parte de persistência do *software Bluetooth*, para extração de dados de simulações. Isto é feito por meio do mapeamento objeto-relacional via JPA (*Java Persistence API* – API Java de Persistência), utilizando uma base de dados relacional HSQL[1] para armazenamento.

7.3 Aplicação de conceitos estudados no curso

A seção 7.2 detalha, para cada disciplina de maior relevância para o trabalho, os conceitos aplicados.

Ainda que não tenham sido listadas, disciplinas fundamentais como Estrutura de Dados e Princípio de Desenvolvimento de Algoritmos também tiveram sua importância direta ou indiretamente.

7.4 Continuidade do trabalho

Considerando continuar atuando na área em que o trabalho foi realizado, um passo importante para aprimorar o sistema seria estender o alcance do mesmo para além do limite físico imposto pelas antenas *Bluetooth*.

A ideia seria que diversas unidades autônomas do *software Bluetooth* poderiam atuar colaborativamente, cobrindo uma mesma área. Essa colaboração seria feita por um canal dedicado (Ethernet, WiFi, etc), não por *Bluetooth*, a fim de que toda a banda disponível do mesmo seja dedicada para suas finalidades.

Dessa maneira, diversos pontos de atuação do *software* poderiam estar separados fisicamente de maneira que o conjunto de pontos seja percebido pelo usuário como se fossem apenas um. Isso possibilitaria, por exemplo, que um espaço com mais de 100m de raio fosse coberto,

[1] http://www.hsqldb.org

ou um ambiente com diferentes cômodos, ou ainda que fosse utilizado apenas para melhorar a qualidade de serviço.

Alguns requisitos de projeto podem estar presentes, como a preocupação com *Single Point Of Failure* (Ponto Central de Falha – SPOF) ou gerenciamento transacional de dados.

Para isso seria necessário um estudo detalhado de sistemas distribuídos, em especial sistemas de arquivos e/ou bancos de dados distribuídos.

Apêndice A

Padrões de projeto utilizados

O livro *Design Patterns: Elements of Reusable Object-Oriented Software*, de Gamma et al. [4] apresenta uma série de padrões de projeto, que são soluções prontas para problemas recorrentes no desenvolvimento de *softwares*.

As seções a seguir mostrarão de que maneira alguns desses padrões foram aplicados tanto nos códigos do *software Bluetooth* quanto no emulador.

A.1 *Adapter*

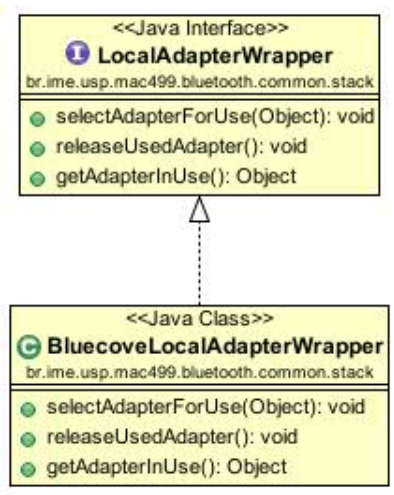


Figura A.1: Padrão *Adapter* para API de escolha de adaptadores *Bluetooth* da BlueCove

Objetivo: "converter a interface de uma classe em outra esperada pelos clientes. O *Adapter* permite que classes trabalhem em conjunto, o que não poderia ocorrer por conta das interfaces incompatíveis" [4].

O uso do *Adapter* no projeto é devido ao fato de a API da JSR 82 não suportar o uso de múltiplos adaptadores *Bluetooth* (pressupõe a existência de um único). No entanto, bibliotecas como a BlueCove permitem que múltiplos adaptadores sejam utilizados, porém por meio de uma API própria.

Para que o código do *software* não fique acoplado diretamente à BlueCove é que foi criado o *Adapter* para a seleção de adaptadores *Bluetooth*. Com isso, caso seja necessário trocar a implementação JSR 82, basta escrever um novo *Adapter* para esta implementação, deixando o código principal livre de alterações (figura A.1).

Outra aplicação do *Adapter* no projeto envolve seleção e inicialização de adaptadores *Bluetooth* (figura A.2). Como a API para seleção de adaptadores é diferente na BlueCove e na BlueCoveEmu, um *Adapter* para cada implementação é responsável por realizar essas tarefas.

Com isso, em tempo de inicialização, basta escolher o *Adapter* correto para o modo de

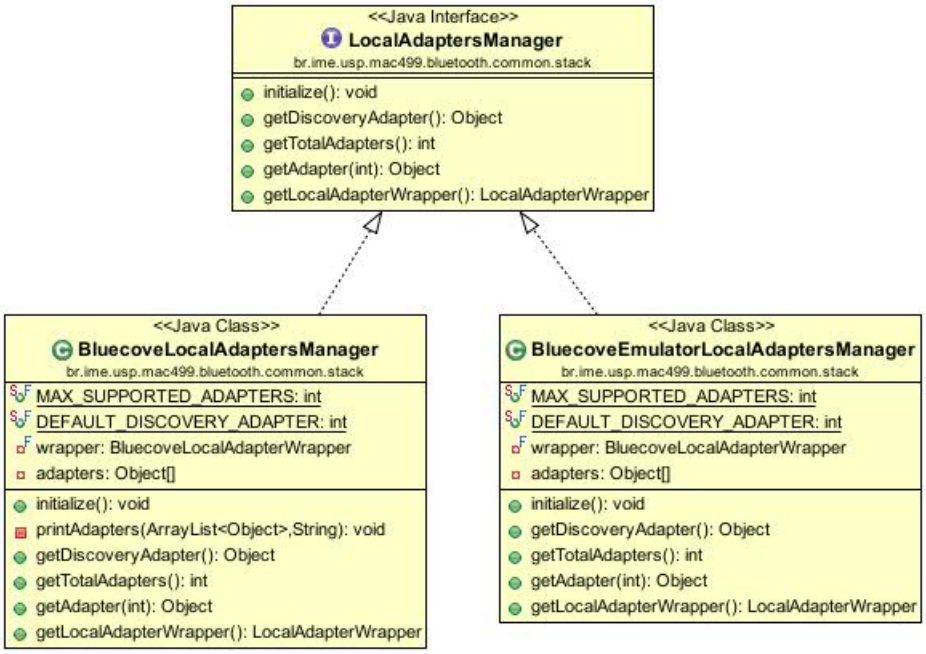


Figura A.2: Padrão *Adapter* para duas APIs de seleção e inicialização de adaptadores *Bluetooth*

operação do *software*.

A.2 *Chain of Responsibility*

Objetivo: "evitar o acoplamento do remetente de uma requisição com o receptor permitindo que mais de um objeto tenha a chance de tratar a requisição. Encadear os objetos receptores e passar a requisição pela cadeia até que um objeto a trate" [4].

No projeto, o padrão é utilizado para que diversos critérios possam ser verificados antes de se proceder ao envio de conteúdos a um dispositivo-alvo como: verificar se um MAC *address* é restrito, se o dispositivo já recebeu o máximo de conteúdos especificados ou verificar qual o próximo conteúdo, caso contrário.

A figura A.3 mostra a *thread* de envio de conteúdos (`RunnableAdvertiser`), a qual possui uma cadeia composta por `AdvertisingDeviceFilterChain`s.

De tempos em tempos a *thread* verifica quais dispositivos estão sob a área de atuação do sistema, e então antes de proceder ao envio de conteúdos a cada um deles, uma filtragem dos que devem ser alvo é realizada por meio da cadeia de filtros.

A.3 *Factory Method*

Objetivo: "definir uma interface para a criação de um objeto, mas deixar as subclasses decidirem que classe instanciar" [4].

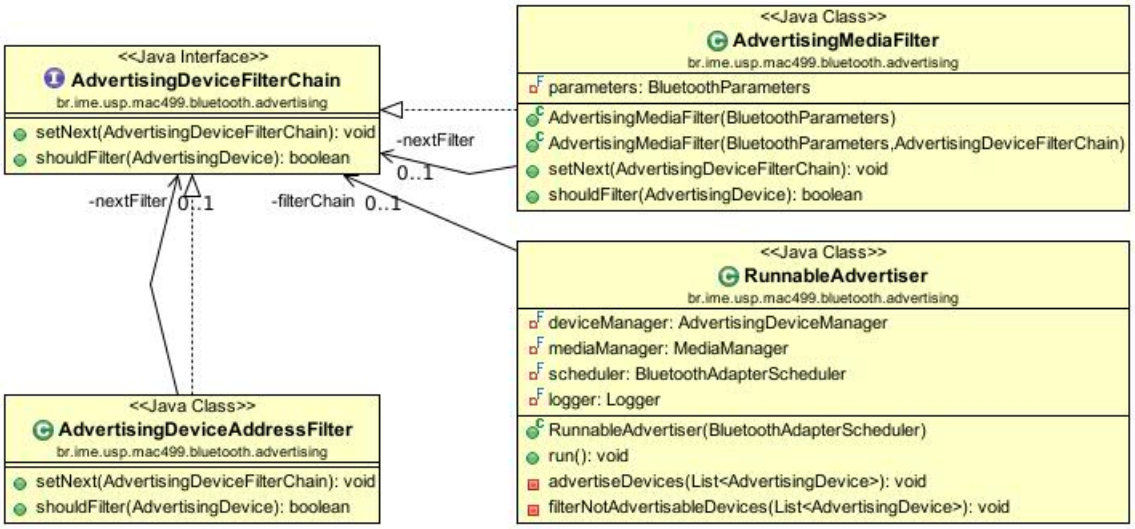


Figura A.3: Padrão *Chain of Responsibility* utilizado para determinar que dispositivos *Bluetooth* devem receber conteúdo no momento

No emulador, dispositivos virtuais *Bluetooth* podem ser criados. Dadas uma série de características, um dispositivo virtual pode ser uma permutação destas. Criar manualmente cada um dos dispositivos pode ser repetitivo, sujeito a erros e o código resultante pode não deixar claras quais as características do dispositivo criado.

Para evitar esses problemas, o padrão é aplicado de maneira parametrizada, onde em apenas uma chamada de método com os parâmetros desejados, obtém-se uma instância com essas características. Exemplos de parâmetros: capacidade de realizar NFC, rejeitar recepção de conteúdos, interromper recepção, ficar invisível ou aleatoriamente visível em processos de descoberta, etc.

A figura A.5 mostra como pode ser composto um `EmulatedAdvertisingDevice` (dispositivo virtual emulado), dados os parâmetros de entrada. Por meio destes, seleciona-se um dispositivo virtual genérico (`GenericEmulatedAdvertisingDevice`) ou uma de suas subclasses com características diversas.

Ainda por meio dos parâmetros faz-se a composição do dispositivo emulado com algum dos manipuladores de requisição OBEX (`ServerRequestHandler`).

A.4 *Strategy*

Objetivo: "definir uma família de algoritmos, encapsular cada um e torná-los intercambiáveis" [4].

Este padrão é aplicado também nos dispositivos virtuais emulados, de maneira que uma chamada a `EmulatedAdvertisingDevice.receivePush()` possa ter resultados diversos como: rejeição do recebimento, interrupção do fluxo de dados, entre outros.

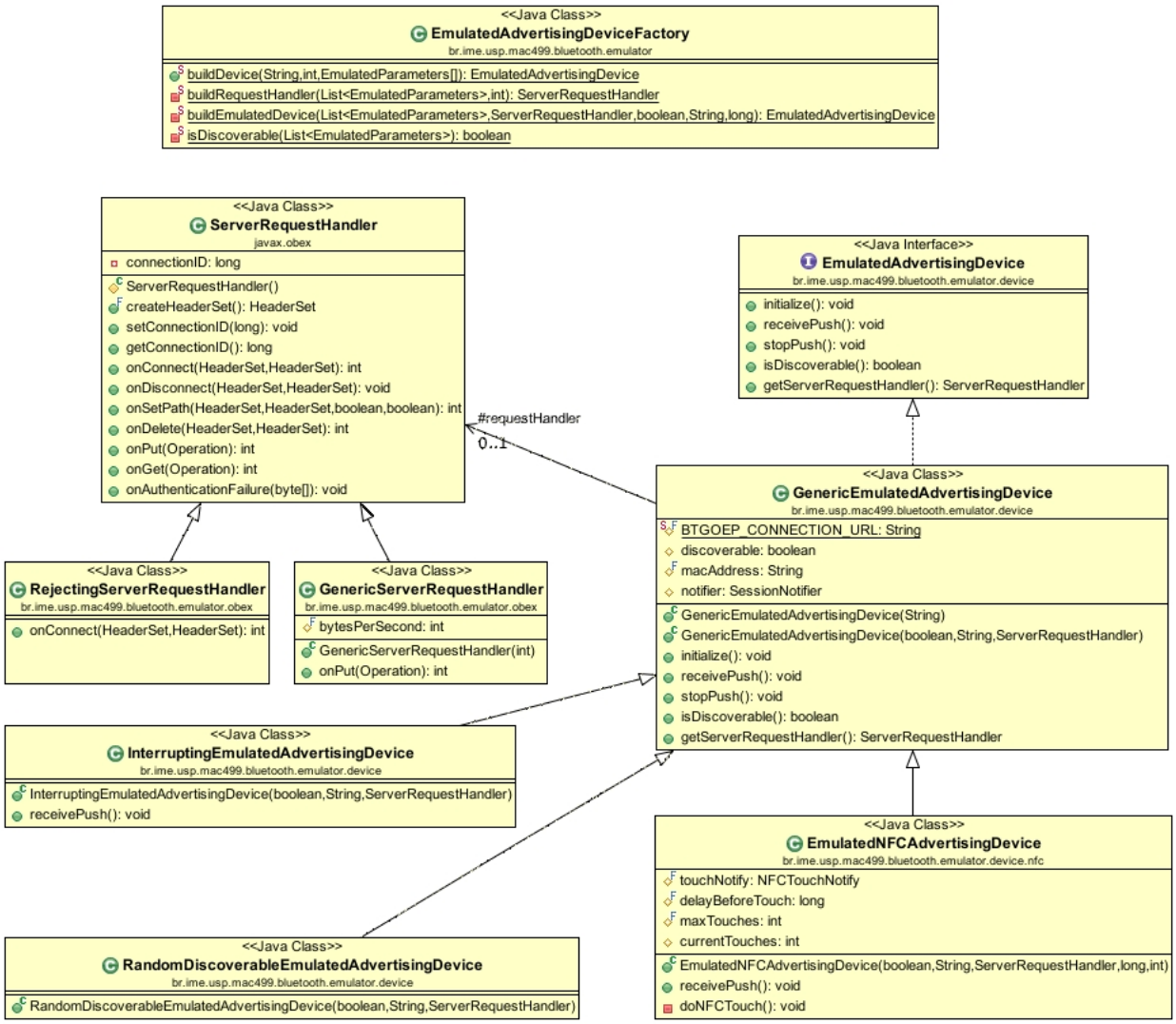


Figura A.4: Padrões *Factory Method* e *Strategy* e envolvidos na criação de um dispositivo *Bluetooth* emulado, o qual pode ter comportamentos variados

A.5 *Observer*

Objetivo: "definir uma dependência um-para-muitos entre objetos de maneira que quando um objeto muda seu estado, todos seus dependentes são notificados e atualizados automaticamente" [4].

No projeto o padrão é utilizado especialmente para verificar novas informações ao final da execução de processos de descoberta de dispositivos e busca de serviços nos mesmos.

A figura A.5 mostra dois observadores do processo de descoberta. O `DiscoveryResultProcessor` é responsável por determinar se um dispositivo descoberto no processo atual já foi detectado em processos passados e assim atualizar suas respectivas informações.

Já o `ServiceSearchDispatcher` dispara uma busca de serviços em um dispositivo que tenha sido descoberto pela primeira vez.

Já a figura A.6 mostra o observador `ServiceSearchProcessor`, que ao fim de cada processo

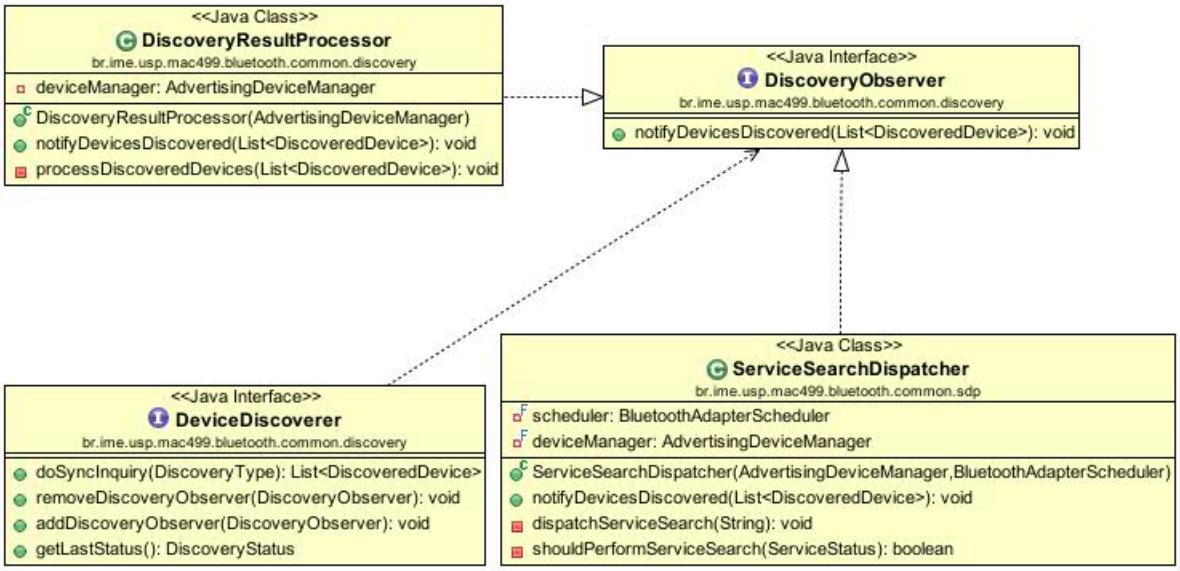


Figura A.5: Padrão *Observer* notificando processadores de resultado de busca de dispositivos

de busca de serviços em um dispositivo, verifica se a mesma foi realizada com sucesso e quais serviços foram detectados, atualizando as estruturas de dados do sistema de acordo.

A.6 *Singleton*

Objetivo: "assegurar que uma classe tem apenas uma instância, e prover um ponto de acesso global a ela" [4].

O padrão é utilizado para evitar que classes que realizem processos de descoberta possuam mais de uma instância, o que poderia causar problemas se processos de descoberta fossem deflagrados simultaneamente.

Com uma única instância de `GenericDeviceDiscoverer` (figura A.7), por exemplo, a mesma é capaz de evitar requisições múltiplas.

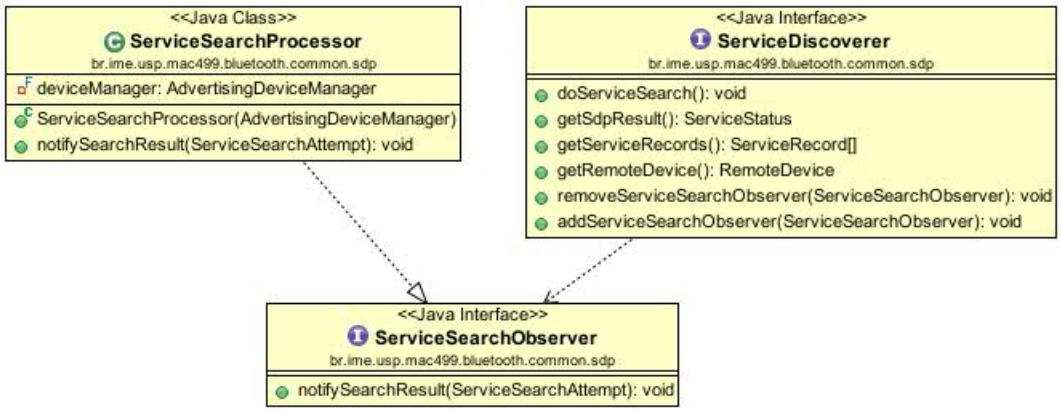


Figura A.6: Padrão *Observer* notificando processadores de resultado de busca de serviços em dispositivos *Bluetooth*

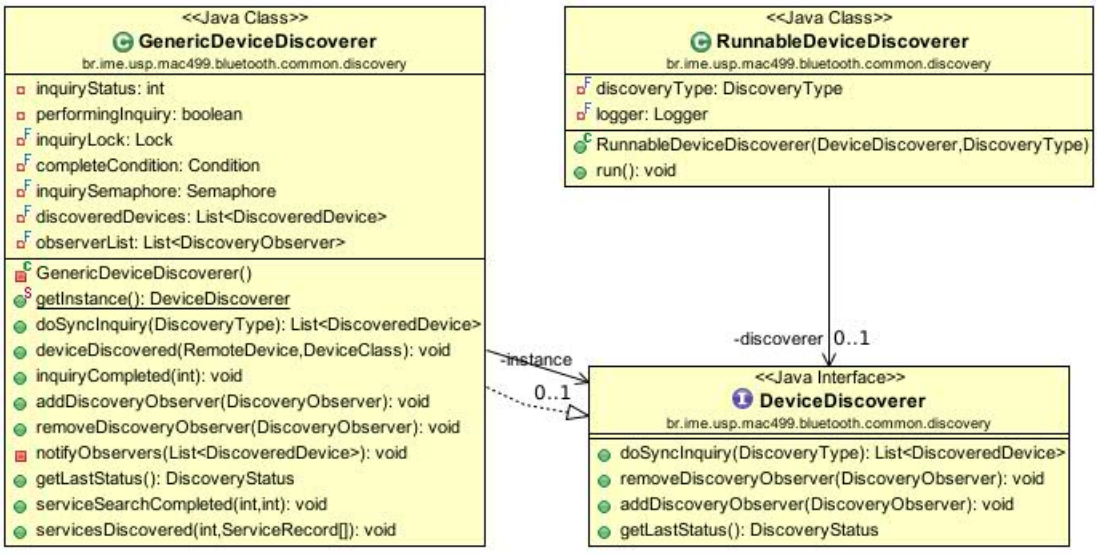


Figura A.7: Padrão *Singleton* utilizado para garantir que haverá apenas um processo de busca de dispositivos no sistema

Apêndice B

Utilização do *software*

O *software* é composto de duas partes principais: o transmissor de conteúdos e modo de emulação.

O transmissor de conteúdos é o sistema de *Bluetooth marketing*. Quando ativado o modo de emulação, o mesmo deixa de utilizar adaptadores *Bluetooth* reais, funcionando apenas com dispositivos virtuais, ambiente no qual podem-se definir cenários de simulação.

A versão da biblioteca BlueCove fornecida, compilada com o *patch* citado em 5.3.1 funciona apenas no sistema Linux (vide tabela 5.1). Em modo de emulação o *software* executa em qualquer sistema.

B.1 Diretórios

O transmissor de conteúdos encontra-se no diretório `software`, enquanto que a parte correspondente ao modo de emulação localiza-se em `software-emulator`. O diretório `software-jsr257` abriga a implementação RMI da JSR 257.

De maneira geral cada um dos projetos contém os seguintes diretórios:

- `config`: arquivos de configuração específicos do projeto;
- `database`: base de dados HSQL, específica para o transmissor ou para o modo de emulação;
- `lib`: bibliotecas do projeto;
- `logs`: arquivos de *log*;
- `media`: conteúdos multimídia;
- `resources`: documentação ou código-fonte de projetos auxiliares;
- `scripts`: *scripts* para execução do projeto;
- `src`: código-fonte do projeto;
- `test`: código-fonte de testes do projeto;

- `uml`: diagramas UML do projeto.

O diretório do modo de emulação possui ainda o diretório `simulation`, o qual contém os cenários de simulação utilizados nas simulações do capítulo 6.

B.2 Arquivos de configuração

O transmissor de conteúdos e modo de emulação possuem arquivos específicos de configuração, que serão descritos a seguir.

B.2.1 Parâmetros do transmissor (`bluetooth.properties`)

O sistema de *Bluetooth marketing* pode ser configurado por meio do arquivo `bluetooth.properties` no diretório `config` do *software*.

- **Intervalo de descoberta** (`discovery.interval`)
  Determina de quanto em quanto tempo deve ser realizado um processo de descoberta de dispositivos, em segundos.
- **Transferências concorrentes em um adaptador** (`bluetooth.max_concurrent`)
  Cada adaptador *Bluetooth* suporta no máximo 7 transferências concorrentes. Este parâmetro permite baixar o número de transferências concorrentes a fim de aumentar a largura de banda disponível para cada dispositivo.
- **Adaptador de descoberta** (`bluetooth.discovery_adapter`)
  Um dos adaptadores *Bluetooth* deve ser eleito para realizar a descoberta, exclusivamente. A escolha é arbitrária, e normalmente depende do melhor posicionamento do adaptador, a fim de obter melhores resultados em descobertas.
- **Tentativas de reenvio** (`advertising.max_retries`)
  Número de vezes em que se deve tentar reenviar conteúdo multimídia a um dispositivo que o rejeita ou apresenta erros/interrupção durante a transmissão.
- **Limite de conteúdos aceitos** (`advertising.max_successes`)
  Número de conteúdos que um dispositivo pode receber com sucesso.
- **Intervalo de verificação para anúncio** (`advertising.interval`)
  Intervalo, em milisegundos, no qual o sistema deve verificar por novos dispositivos a receberem conteúdos multimídia. Esses dispositivos são provenientes da execução de sucessivos processos de descoberta.

B.2.2 Filtros (`filter.properties`)

O arquivo `filter.properties` no diretório `config` permite que testes sejam feitos com dispositivos reais sem que todos os dispositivos *Bluetooth* da vizinhança participem dos mesmos involuntariamente.

```
filter.mode=do_filter

# Allow my mobile
allowed.0=001122334455

# Avoid bothering dad
blacklisted.1=554433221100

...

# Nth device
allowed.N=XXXXXXXXXXXX
```

Figura B.1: Formato geral do arquivo de filtros `bluetooth.properties`

A figura B.1 ilustra o formato geral do arquivo.
O filtro possui três modos de operação:

- **Permitir todos** (`allow_all`)
  Não realiza qualquer filtragem de dispositivos, mesmo que sejam especificados em seguida no arquivo.
- **Negar todos** (`deny_all`)
  Filtra qualquer dispositivo que apareça na área de atuação. Opção normalmente utilizada para teste do *software*.
- **Filtrar** (`do_filter`)
  Realiza filtragem de dispositivos, permitindo ou filtrando cada um por seu endereço MAC.

O modo `do_filter` permite especificar que dispositivos devem ou não passar pelo filtro da seguinte forma:

- `allowed.X=001122334455`: permite que o dispositivo de endereço MAC `001122334455` passe pelo filtro;
- `blacklisted.Y=554433221100`: impede que o dispositivo de endereço MAC `554433221100` passe pelo filtro, não sendo detectado por processos de descoberta, busca de serviços etc.

A numeração de cada entrada `allowed` ou `blacklisted` deve ser feita sequencialmente de 0 a n.

B.2.3 Modo de emulação (`nfctouch.properties`)

A configuração do leitor NFC emulado localiza-se no arquivo `nfctouch.properties` no diretório `config`.

Atualmente o único parâmetro aceito é `nfc.readers=N`, onde `N` é um número qualquer de leitores NFC emulados.

Vale ressaltar que cada leitor permite que ocorra apenas um toque por vez, assim como nos dispositivos físicos. Se múltiplos dispositivos tentarem tocar o leitor, uma fila de espera é automaticamente criada e os dispositivos vão tocando o leitor em uma ordem arbitrária.

B.3 Opções de linha de comando

O transmissor de conteúdos aceita os seguintes parâmetros de linha de comando:

- `-mediaFile=musica.mp3,foto.jpg`: especifica uma lista de arquivos que podem ser enviados, separados por vírgula.

Já em modo de emulação aceita também os seguintes parâmetros:

- `-adapters=N`: número de adaptadores *Bluetooth* a emular (mínimo 1);
- `-simulationFile=simulation.properties`: caminho para o arquivo do cenário de simulação.

B.4 Base de dados

O diretório `database` armazena os dados correspondentes à execução do modo normal ou emulado no diretório do projeto correspondente.

A URL JDBC `jdbc:hsqldb:file:/path/database/bluetoothnfc` permite consultar o conteúdo dessa base de dados utilizando um driver HSQL. O `path` deve ser substituído pelo caminho absoluto ao diretório onde está a base de dados.

B.5 Logs

Os *logs* de execução do sistema ficam na pasta `log` de cada um dos projetos e são os seguintes:

- `discovery.log`: registra a execução de cada processo de descoberta, número de dispositivos encontrados, duração do processo;
- `sdp.log`: registro da execução de cada processo de busca de serviços em cada dispositivo, bem como o resultado da busca;
- `advertising.log`: mostra o processo de tentativa de veiculação de anúncios para os dispositivos *Bluetooth*, com início, fim e resultado.

Os *logs* são automaticamente arquivados a cada 24 hs, portanto em execuções do *software* em dias diferentes, um *timestamp* é adicionado ao final de cada arquivo de *log* mais antigo.

B.6 Compilação

Para compilar, basta utilizar o Apache Ant ou o Eclipse.

No diretório de cada projeto, digitar `ant` compila os arquivos necessários.

B.7 Execução

Junto com os *softwares* desenvolvidos são fornecidos *scripts* para execução dos mesmos em modo normal ou emulado.

Os *scripts* localizam-se no diretório `scripts` do projeto principal e são os seguintes:

- `run-mac499.sh`: inicia o *software* em modo normal;
- `run-mac499-emu.sh`: inicia o *software* em modo emulado, deve ser fornecido um arquivo de simulação.

Cada um desses *scripts* aceita as opções de linha de comando especificadas em B.3.

```
user@desktop:~# run-mac499.sh -mediaFile=/home/sep/palmeiras.jpg
```

Figura B.2: Exemplo de execução do *software* em modo normal

```
user@desktop:~# run-mac499-emu.sh -adapters=4
-simulationFile=/home/sep/simulation.properties -mediaFile=/home/sep/hino.mp3
```

Figura B.3: Exemplo de execução do *software* em modo emulado

Figura B.4: O paradoxo da comunicação sem fios: teste de funcionalidade com 16 adaptadores *Bluetooth*

Referências Bibliográficas

[1] G.R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 1999.

[2] Myra Dideles. Bluetooth: a technical overview. *Crossroads*, 9(4):11–18, 2003.

[3] Patrick Murphy Erik Welsh and Patrick Frantz. Improving connection times for bluetooth devices in mobile environments. Disponível em http://scholarship.rice.edu/bitstream/handle/1911/20443/Wel2002Mar5ImprovingC.PDF?sequence=1, Março 2002.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, 1995.

[5] Ecma International. Near field communication white paper. Disponível em http://www.ecma-international.org/activities/Communications/2004tg19-001.pdf, 2004.

[6] J. de Nes M. Aiello, R. de Jong. Bluetooth broadcasting: How far can we go? an experimental study. In *Int. Conference on Pervasive Computing and Applications (ICPCA'09) IEEE Computer*, page 2, 2009.

[7] Java Community Process. Jsr 257: Contactless communication api. Disponível em http://jcp.org/en/jsr/summary?id=257, Outubro 2006.

[8] Yaniv Shaked and Avishai Wool. Cracking the bluetooth pin. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 39–50, New York, NY, USA, 2005. ACM.

[9] Bluetooth SIG. About the bluetooth sig. Acessado em 9/5/2010, http://www.bluetooth.com/English/SIG/Pages/default.aspx.

[10] Bluetooth SIG. Specification of the bluetooth system core v2.1 + edr. Disponível em http://www.bluetooth.com/SpecificationDocuments/Core_V21__EDR.zip, Julho 2007.