Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Programa de Pós-graduação em Engenharia de Eletricidade

Antonio Sá Fernandes Palmeira Filho

SNU: um framework para o desenvolvimento de aplicações voltadas às redes ad-hoc espontâneas

São Luís
2007

Antonio Sá Fernandes Palmeira Filho

SNU: um framework para o desenvolvimento de aplicações voltadas às redes ad-hoc espontâneas

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, como requisito parcial para a obtenção do grau de MESTRE em Engenharia de Eletricidade.

Orientador: Francisco José da Silva e Silva

Doutor em Ciência da Computação – UFMA

São Luís

2007

Palmeira Filho, Antonio SF

SNU: um framework para o desenvolvimento de aplicações voltadas às redes ad-hoc espontâneas / Antonio SF Palmeira Filho. – São Luís, 2007.

105 f.

Dissertação (Mestrado) – Universidade Federal do Maranhão – Programa de Pós-graduação em Engenharia de Eletricidade.

Orientador: Francisco José da Silva e Silva.

1. Sistemas de Computação. 2. Computação Móvel. 3. Redes Ad-hoc. I. Título.

CDU 004.75

Antonio Sá Fernandes Palmeira Filho

SNU: um framework para o desenvolvimento de aplicações voltadas às redes ad-hoc espontâneas

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Antonio Sá Fernandes Palmeira Filho e aprovada pela comissão examinadora.

Aprovada em 01 de outubro de 2007

BANCA EXAMINADORA

Francisco José da Silva e Silva (orientador)

Doutor em Ciência da Computação – UFMA

Markus Endler

Doutor em Informática – PUC/Rio

Zair Abdelouahab

Ph. D. em Ciência da Computação – UFMA

Aos meus pais, irmãos e minha esposa, Lorena.

Resumo

Dispositivos móveis evoluíram de simples agendas, calendários ou telefones celulares para modernos assistentes pessoais, capazes de armazenar e processar áudio e vídeo em tempo real. Contudo, dar a melhor utilidade a esta tecnologia em nossas tarefas do dia-a-dia ainda é um desafio. A grande variedade de dispositivos, a crescente necessidade de serviços móveis pervasivos e a heterogeneidade das tecnologias de rede são apenas alguns exemplos dos problemas possivelmente encontrados pelos desenvolvedores ao produzir *software* para o mundo móvel.

Com a popularização das tecnologias de rede, é cada vez mais comum encontrarmos dispositivos móveis capazes de estabelecer conexões através de diferentes tecnologias. Redes infra-estruturadas disponibilizam boas larguras de banda e baixa latência, porém necessitam de uma infra-estrutura de pontos de acesso nem sempre disponível. Por outro lado, as redes *ad-hoc* não necessitam de infra-estrutura e a comunicação entre seus nós é feita ponto a ponto, razão pela qual desempenham um importante papel nos ambientes formados por dispositivos móveis. Redes espontâneas são redes *ad-hoc* formadas ao acaso por usuários que desejam participar de alguma tarefa colaborativa, como o compartilhamento de arquivos.

Neste trabalho apresentamos um *framework* cujo objetivo é ajudar os desenvolvedores a criar facilmente aplicações capazes de gerenciar e compartilhar conteúdo digital com grupos de usuários em redes *ad-hoc* espontâneas de curto alcance. São descritos a arquitetura do *framework* proposto, seus serviços e relacionamentos, assim como alguns cenários de uso, aplicações e trabalhos relacionados.

Palavras-chaves: Computação Móvel. Pervasiva. Ubíqua. *Ad-hoc. Framework.*

Abstract

Mobile devices have evolved from simple personal digital assistants or cell phones to cutting-edge personal digital assistants, aliasing or combining on-the-fly video/audio acquisition and data processing. But giving the best usage to all this new technology in our day-by-day tasks still is challenging. The large variety of devices, the increasing need for ubiquitous mobile services and the network heterogeneity are just some problems developers may find when deploying for the mobile world.

With the popularization of the wireless network technologies, we can find devices capable of establishing network connections using different technologies. Infrastructured networks support good bandwidth with low latency, but they demand an infrastructure of access points that are not available at any place. On the other hand, ad-hoc networks don't need an infrastructure and the communication is done in a peer-to-peer maner. The ad-hoc networks play an important rule in the environments made by mobile devices since they ensure high availability. Spontaneous networks are ad-hoc networks formed occasionally by users that wish to engage in some collaborative task, like a file sharing.

In this work we describe a framework to help developers to easily create applications capable of managing and sharing digital content within groups of people in a spontaneous short-range ad-hoc network. We describe the framework architecture, its services and their relationship, as well as some usage scenarios, applications and related works.

Key-words: Mobile computing. Pervasive. Ubiquitous. Ad-hoc. Framework.

Agradecimentos

À Deus, pela existência. Aos meus pais, Antonio e Conceição, pela crença e perseverança em seus filhos. A Zélia Serra, minha avó materna, Josélia Viana, minha madrinha e Ruberval Palmeira, que sempre se fizeram presentes em toda minha vida. A Rodrigo e Lucas, pelo carinho e admiração que só existe entre irmãos.

À minha esposa, Lorena Etienne, pela consideração e reconhecimento dos muitos momentos roubados do nosso convívio em virtude deste trabalho. Sua compreensão foi decisiva.

Ao meu orientador Francisco, pelo acompanhamento sempre presente e pelas lições e experiências compartilhadas que sem dúvidas contribuíram na minha formação acadêmica e pessoal.

A Pablo Durans, Jack Oliveira e Egídio de Carvalho Júnior, membros do Laboratório de Sistemas Distribuídos, pelo apoio e envolvimento neste projeto.

"Se as pessoas são boas só por temerem o castigo e almejarem uma recompensa, então realmente somos um grupo muito desprezível."

Albert Einstein

Sumário

Lista de Figuras

Lista de Tabelas

Listagens

1 Introdução

Atualmente é inegável que os telefones móveis são muito mais do que apenas dispositivos de comunicação pessoal. Eles evoluíram de simples dispositivos transmissores de voz para assistente digitais, mais parecidos com PDAs (*Personal Digital Assistant*), capazes de adquirir, processar e armazenar grandes quantidades de dados - através de cameras digitais, microfones, gps, dentre outros dispositivos integrados - assim como compartilhá-los através de conexões de rádio de curto alcance ou até mesmo através da própria Internet. A natureza pessoal e a crescente disponibilidade de telefones celulares e PDAs de baixo custo, porém poderosos, nos permite vislumbrar ambientes pervasivos onde possamos interagir e explorar suas vantagens.

Entretanto, para que sejam de fato úteis, os sistemas móveis necessariamente precisam apoiar e facilitar várias atividades humanas e oferecer funcionalidades simples e fáceis. Hoje em dia, telefones móveis são capazes de reproduzir diversos conteúdos, como vídeos, áudios e textos. Porém, utilizá-los para compatilhar tais recursos em ambientes colaborativos ainda não é uma tarefa simples. Dispositivos sofisticados têm a capacidade de enviar e receber conteúdos utilizando a tecnologia *Bluetooth* [Met99], uma conexão de rádio de curto alcance bem definida. Escolher um dado e enviá-lo via *Bluetooth* é trivial. Entretanto, é também desejável que dispositivos móveis sejam capazes de prover um leque de serviços que auxiliam as interações sociais [Bea05]. Quase sempre disponíveis, estes dispositivos altamente pessoais podem se tornar uma ferramenta importante, ajudando as pessoas no compartilhamento de suas experiências e permitindo que colegas de trabalho possam coordenar seus afazeres.

A capacidade de formarem redes utilizando tecnologias diversas às utilizadas para comunicação de voz abre um novo horizonte de possibilidades. Usuários podem estabelecer conexões de rádio de curto alcance com a mesma facilidade com que fazem ou atendem uma ligação telefônica. Bater uma fotografia e em seguida enviá-la via *Bluetooth* ou infravermelho [Ass, Lab] é uma tarefa cada vez mais comum entre os usuários.

Entre as tecnologias de rede sem fio, tem-se observado um crescente interesse

por redes sem infra-estrutura ou MANETs, mais comumente conhecidas como redes *ad-hoc*. Denominam-se WPAN, *Wireless Personal Area Network*, as redes pessoais sem fio que provêem conectividade entre nós relativamente próximos, a uma distância máxima de 10 metros. A tecnologia de rede pessoal sem fio mais popular é a *Bluetooth*, amplamente disponível e estudada. Uma das subclasses de redes pessoais são as redes espontâneas [EH00], formadas por um grupo restrito de pessoas, em um período de tempo pequeno, que se reúnem para alguma atividade colaborativa em comum. Nestas pequenas redes sem infra-estrutura questões de encaminhamento de mensagens não são relevantes e a colaboração é basicamente feita ponto-a-ponto.

O conjunto atual de funções disponível nos dispositivos móveis, especialmente sua capacidade de se comunicar via redes espontâneas, suporta um modelo rico de interação em multimídia, capacitando usuários a compartilharem recursos digitais diretamente a partir dos seus dispositivos móveis [WPD+02]. A seguir, identificamos alguns possíveis cenários onde usuários desempenham tarefas colaborativas utilizando dispositivos móveis e estabelecemos os requisitos para uma solução que torne possível o nível de interação desejado.

1.1 Cenários e Requisitos

Imagine que participamos de uma reunião de negócios. Seria muito útil se pudéssemos compartilhar uma coleção de documentos e ao mesmo tempo utilizar um quadro branco para indicar detalhes em uma figura. Talvez fosse útil enviar um vídeo para vários componentes do grupo ao mesmo tempo. Ao final da reunião poder-se-ia utilizar uma agenda comum para marcar novos compromissos.

Estudantes poderiam compartilhar suas anotações de sala, exercícios ou documentos fornecidos pelos professores. Poderiam estabelecer um bate-papo virtual para postar questões e respostas relativas às aulas e armazenar o histórico para estudos futuros. Poderiam ainda trocar idéias [Bea05] entre si, ou utilizar um comunicador ponto-a-ponto, trocando mensagens de texto ou arquivos com amigos individualmente ou com grupos de amigos.

Imagine agora que jovens amigos estão em um bar num encontro casual ou em um aeroporto esperando por aquele vôo de férias. Em ambos os cenários eles desejam

relaxar e procurar por novos amigos. Poderiam localizar pessoas ao redor que possuam a mesma faixa etária ou os mesmos gostos; talvez visualizar alguma informação pública sobre elas, como seu estado cível, profissão ou até mesmo uma fotografia. No caso de encontrar algo interessante, poderiam iniciar um bate-papo via texto de forma anônima, até considerar que vale a pena se conhecer pessoalmente. Poderiam, então, enviar fotografias de sua vizinhança e brincar de "quem sou eu", por exemplo.

Todos estes cenários podem ser satisfeitos com a utilização de um dispositivo móvel capaz de localizar e se comunicar com semelhantes ao seu redor. Para tanto, é interessante que haja um *framework* capaz de fornecer um conjunto de serviços sobre os quais uma aplicação colaborativa poderia ser construída de forma rápida e eficiente.

A partir destes cenários, este trabalho identificou os seguintes requisitos funcionais para o *framework* em questão:

- Inexistência de um controle centralizado - o *framework* deve ser direcionado para redes espontâneas *ad-hoc* de curto alcance, onde componentes colaboram de forma espontânea e sem a necessidade de um gerenciador centralizado de serviços, garantindo assim que possa ocorrer o compartilhamento aonde quer que estejam os usuários;

- Consumir poucos recursos - a comunicação entre dispositivos deve ser rápida para que haja economia de energia. Deve existir persistência das informações gerenciais (contatos e grupos) assim como nos dados compartilhados de forma simples e eficiente, sem um consumo exagerado de memória de massa, disponibilizando o máximo dos recursos de armazenamento para os dados do usuário;

- Independência da tecnologia de rede - *Bluetooth* é a tecnologia *ad-hoc* mais difundida, contudo existem outras tecnologias *ad-hoc* como a especificação 802.11 [Ass03]. Portanto é desejável apresentar uma solução que proveja independência da tecnologia de rede em uso e assim permita acesso amplo aos diversos dispositivos móveis disponíveis;

- Portabilidade da solução - para que haja uma maior aceitação da solução proposta, se faz necessária a utilização de uma plataforma amplamente disponível e portada para os diversos dispositivos móveis existentes.

1.2 Objetivos

Esta pesquisa tem por objetivo geral o desenvolvimento de um *framework* que facilite a construção de aplicações voltadas ao compartilhamento de conteúdo em redes *ad-hoc* espontâneas de curto alcance. Os objetivos específicos deste trabalho compreendem:

- Estudo do estado da arte em frameworks para compartilhamento de conteúdo em ambientes de computação móvel;
- Definição de uma arquitetura para o framework que atenda aos requisitos identificados na seção 1.1. Ressalta-se a importância da inexistência de serviços ou controles centralizados, dado que o trabalho é focado em redes *ad-hoc* espontâneas;
- Implementação da arquitetura proposta;
- Avaliação do framework, a ser constituída de medições que analisem a aderência aos requisitos estabelecidos e sua efetividade em simplificar o processo de desenvolvimento de *software* voltados à redes espontâneas. Deve-se também verificar a aplicabilidade do framework a uma variedade de aplicações, de forma que o mesmo possa ser utilizado em diversos cenários.

1.3 Estrutura da Dissertação

Este trabalho descreve a arquitetura do SNU, sua implementação e avaliação. O Capítulo 2 apresenta a base teórica necessária para o entendimento dos problemas e dificuldades envolvidos neste trabalho e em seguida, o Capítulo 3 discute relevantes trabalhos relacionados. O Capítulo 4 descreve a arquitetura do SNU, seus serviços e detalhes de implementação. O Capítulo 5 traz uma análise dos recursos utilizados pelo mesmo, uma avaliação de desempenho dos diversos serviços apresentados e algumas aplicações construídas com o auxílio do SNU. O Capítulo 6 apresenta as conclusões obtidas a partir deste trabalho além de apresentar os trabalhos futuros que podem ser desenvolvidos a partir deste esforço inicial.

Todo este projeto e sua documentação está disponível livremente sob a licença LGPL (*Lesser General Public License*) no endereço eletrônico `http://lsd.ufma.`

br/~snu/. Para maiores detalhes sobre a licença LGPL, acesse o endereço eletrônico http://www.gnu.org/licenses/lgpl.html.

2 Computação Móvel: Princípios e Desafios

Entende-se por computação móvel aquela realizada em dispositivos de computação portáteis capazes de acessar uma rede sem fio. Relacionado com a computação móvel, alguns termos são comumente utilizados [HMNS03]: computação nômade, formada por usuários que se movimentam, porém realizam suas tarefas em local fixo ou com limitada mobilidade (ex. linhas discadas); computação ubíqua, disponibiliza computadores através de um espaço físico, tornando-os invisíveis aos usuários (computadores tornam-se parte da nossa vida de forma transparente); e computação pervasiva, onde há acesso conveniente a informações relevantes e a habilidade de realizar ações sobre as mesmas quando e onde for necessário.

O advento da computação móvel possibilitou o desenvolvimento de uma computação mais próxima do usuário, capaz de interagir com seu meio ambiente, disponibilizando funcionalidades antes confinadas aos computadores de mesa. Este capítulo apresenta uma breve introdução à computação móvel, iniciando pelos seus principais princípios para, em seguida, apresentar os novos desafios impostos por essa tecnologia.

2.1 Princípios da Computação Móvel

Ambientes de computação móvel podem ser entendidos como um novo estágio na evolução da computação distribuída. PDAs e *Smartphones*, por exemplo, podem trabalhar de forma autônoma ou como uma extensão de um sistema maior, projetado para administrar inúmeros dispositivos remotos. Estes ambientes são caracterizados por diversos atributos que serão vistos mais detalhadamente nas subseções a seguir.

2.1.1 Descentralização

A computação móvel é um passo além na descentralização promovida pelos sistemas distribuídos. Uma grande variedade de pequenos dispositivos cooperam estab-

elecendo uma rede dinâmica de relacionamentos. A habilidade de utilizar aplicações e informações em dispositivos móveis gera a necessidade de sincronização de atualizações de dados com sistemas servidores e outros dispositivos.

Dispositivos e aplicações são comumente parte de um sistema infra-estruturado, como uma rede de celular. A descentralização torna obrigatória a administração dos dispositivos e aplicações remotas, a fim de prover atualizações específicas para cada dispositivo. Os serviços de administração de sistemas descentralizados devem conhecer as especificações de cada dispositivo vinculado ao sistema, assim como conhecer os perfis de uso dos seus usuários, desta forma podem determinar o melhor nível de interação entre usuários e sistemas. Tais requisitos exigem um alto grau de escalabilidade e flexibilidade das aplicações, pois estes sistemas podem administrar milhões de dispositivos móveis ao redor do mundo (*roaming*), ao invés de apenas alguns milhares de computadores de mesa.

2.1.2 Diversificação

Dispositivos móveis em geral apresentam soluções direcionadas a problemas específicos, compostas por *softwares* e *hardwares* em geral propietários. Usuários podem utilizar alguns dispositivos em conjunto, que talvez tenham funções em comum, mas que foram desenhados para solucionar problemas bem definidos (ex. celular e PDA, ambos podem possuir câmeras fotográficas, porém são dispositivos com propósitos diferentes). Computadores de mesa seguem um padrão de *hardware* (ex. IBM-PC i386) e, portanto, podem ser utilizados em diversos cenários. Contudo, dispositivos móveis são direcionados para contextos específicos e utilizam o melhor conjunto de *software* e *hardware* para sua solução, sem necessariamente se ater a padrões universais previamente estabelecidos.

Uma grande barreira para o desenvolvimento de aplicações móveis é a diversidade de especificações e funções encontradas nos dispositivos [IF03]. Como não há um padrão único, cada fabricante utiliza a solução que melhor lhe convier, tornando difícil o desenvolvimento de aplicações comuns. A interface com o usuário é provavelmente a mais clara diferença entre os dispositivos móveis. Alguns possuem visores coloridos de alta resolução enquanto outros são capazes de exibir apenas textos. Aplicações desenvolvidas para visores pequenos não devem ser apenas "esticadas" para preencher todo o espaço disponível em visores maiores, porém precisam ser redesenhadas a fim de prover um maior grau de usabilidade.

Alguns dispositivos utilizam teclados ou apontadores em visores sensíveis ao toque, outros já possuem reconhecimento de voz. Capacidade de processamento, memória física e de massa, autonomia (capacidade da bateria) e conectividade são outros pontos comumente divergentes entre os dispositivos.

2.1.3 Simplicidade

Apesar da inquestionável utilidade de computadores em geral, eles exigem treinamento específico e muitas das vezes demorado. Por outro lado, sistemas pervasivos são tipicamente específicos e devem ser simples de usar. Contudo, ser simples não significa ser primitivo: telas sensitivas ao toque, reconhecimento da escrita ou da voz, segurança a partir de dados biométricos (ex. digital do polegar) são funcionalidades tipicamente empregadas nos dispositivos móveis atuais.

Usuários de dispositivos móveis tendem a utilizá-los sem treinamento prévio e, portanto, esperam interfaces intuitivas, rápidas e versáteis. Porém, quanto mais sofisticada é a interface com o usuário, maior o poder computacional necessário para seu funcionamento. Em geral, maior poder computacional significa mais consumo de energia e mais espaço físico, ou seja, equipamentos maiores. Entretanto, dispositivos móveis devem ser pequenos e leves, consumir pouca energia para efetuar suas tarefas básicas e apresentar interfaces rápidas e intuitivas. Equacionar usabilidade, recursos e dimensões é um dos principais desafios dos atuais fabricantes de dispositivos móveis [IF03].

2.1.4 Conectividade

Uma grande variedade de tecnologias de comunicação sem fio está disponível atualmente: redes celulares, WLANs, *Bluetooth*, IrDA, GPRS/EDGE/3G, etc. Cada tecnologia possui características próprias, variando diversas propriedades como largura de banda, latência, abrangência geográfica, consumo de energia dentre outras. Como forma de agrupá-las, utilizaremos o alcance do enlace de rádio, exemplificado na Figura 2.1, a saber:

- **WPAN - Wireless Personal Area Network (10m)**
  Bluetooth (IEEE 802.15.1), UWB (Ultrawideband, IEEE 802.15.3), ZigBee (IEEE

802.15.4), RFID (Radio Frequency Id, etiquetas inteligentes) e NFC (Near Field Communications). As WPANs são redes de curto alcance, baixa largura de banda (10Mbps), que não necessitam de infra-estrutura nem de licença governamental para operar. São voltadas para a transferência de pequenos blocos de dados, em comunicações ponto-a-ponto onde não há a necessidade de alta confiabilidade e disponibilidade.

- **WLAN - Wireless Local Area Network (150m)**
  WiFi a/b/g (IEEE 802.11 a/b/g), IEEE 802.11n, IEEE 802.11s - As WLANs são redes de médio alcance, razoável largura de banda (100Mbps), que necessitam de infra-estrutura (mas podem ser também usadas no modo *ad-hoc*), porém não precisam de licença governamental para operar. Podem ser utilizadas em sistemas que necessitem de boa confiabilidade e disponibilidade.

- **WMAN - Wireless Metropolitan Area Network (50km)**
  WiMax (IEEE 802.16 a/d/e), WiBRO, Móbile Fi (IEEE 802.20). As WMAN são redes de longo alcance, razoável largura de banda (50Mbps), que necessitam de infra-estrutura e de licença governamental para operar. Podem disponibilizar acesso a cidades inteiras e podem ser utilizadas em sistemas que necessitem de uma conexão sempre ativa, com boa confiabilidade e disponibilidade. Sofrem influência do meio ambiente, variando sensivelmente sua largura de banda e latência.

- **WWAN - Wireless Wide Area Network (continental)**
  CDMA(1x Rtt, 1x EV-DO, 1x EV-DOa), OFDM, TDMA, GSM/GPRS, EDGE, UMTS, HSDPA, HSUPA. As WWAN são redes de longo alcance, baixa largura de banda (1Mbps), que necessitam de infra-estrutura e de licença governamental para operar. Podem disponibilizar acesso continental, ou até mesmo em toda a superfície terrestre. Podem ser utilizadas em sistemas que necessitem de uma conexão sempre ativa, com razoável confiabilidade e disponibilidade. Contudo, sofrem grande influência do meio ambiente e por isso podem apresentar grande variação de largura de banda e latência.

Redes Infra-estruturadas, *Ad-hoc* e Espontâneas

As redes sem fio podem ser dividas em infra-estruturadas e *ad-hoc*. Infra-

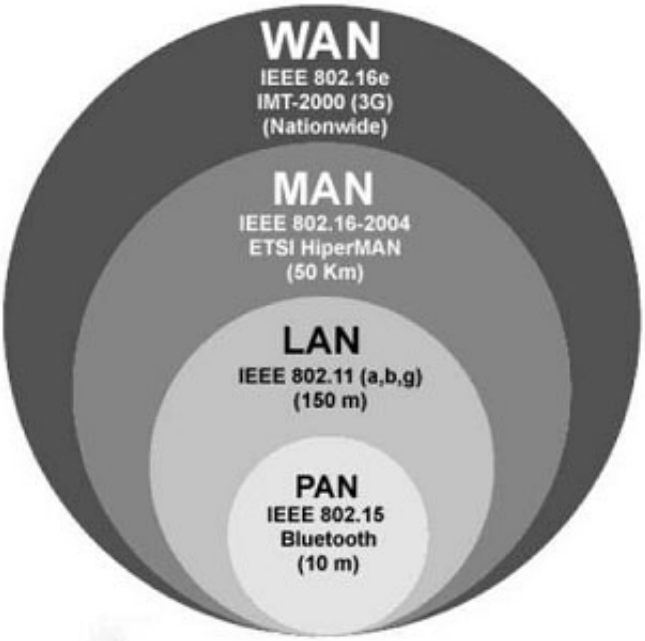


Figura 2.1: Diagrama das redes sem fio

estruturadas são redes onde há comunicação entre nós móveis (computadores portáteis) e os pontos de acesso. Os pontos de acesso são transceptores (transmissor/receptor) conectados a rede local Ethernet convencional (com fio) responsáveis por conectar os diversos dispositivos móveis à mesma. As redes infra-estruturadas utilizam um projeto simplificado, uma vez que são baseadas em rotas conhecidas e confiáveis para o encaminhamento das mensagens. Porém, perdem em flexibilidade já que podem não estar disponíveis em caso de desastre (onde há a destruição da infra-estrutura de suporte da rede).

As redes *ad-hoc* não necessitam de infra-estrutura e sua comunicação é feita diretamente entre os nós móveis (ponto-a-ponto), que possuem uma maior complexidade já que todos devem implementar mecanismos de acesso ao meio. São mais robustas e podem estar disponíveis em caso de catástrofes. As redes *ad-hoc* utilizam projetos mais complexos, pois, como os dispositivos não possuem um ponto comum de conexão a rede (um ponto de acesso), não há uma rota preestabelecida para o caminhamento das mensagens. Os caminhos possíveis entre os diversos nós da rede devem ser descobertos e mantidos por todos os nós, sempre analisando possíveis alterações causadas pelas entradas e saídas de novos nós [AGIS00].

Dispositivos móveis como celulares e PDAs capazes de estabelecer uma conexão *Bluetooth* podem formar uma WPAN *ad-hoc* e assim desenvolver tarefas cooperativas sem

a necessidade de uma infra-estrutura de suporte. Caso os usuários destes dispositivos se encontrem de forma aleatória e imprevisível e formem redes de topologias variáveis e temporárias, encontraremos as ditas redes espontâneas [PGGH03]. Portanto, quando amigos que a muito não se viam, se encontram por acaso em um restaurante e compartilham fotos utilizando seus *Smartphones* via *Bluetooth*, estão de fato utilizando uma rede *ad-hoc* espontânea de curto alcance.

2.2 *Bluetooth*

Por se tratar de uma tecnologia de baixo custo e grande disponibilidade, o *Bluetooth* foi eleito como a tecnologia de comunicação sem fio inicial para o desenvolvimento desta pesquisa. Sendo assim, esta seção apresenta uma breve descrição desta tecnologia, trazendo suas características, capacidades, protocolos, vantagens e desvantagens.

A tecnologia *Bluetooth* é comumente utilizada para conectar os mais variados dispositivos como celulares, PDAs, fones de ouvido, microfones, computadores e teclados. Pode ser utilizado para estabelecer conexões sempre ativas, como aquelas utilizadas pelos fones de ouvido sem fio, ou para transferências rápidas como o compartilhamento de um toque musical de celular.

Bluetooth pode ser utilizado nos mais diversos cenários: no escritório, para sincronizar um PDA com um computador de mesa bastando-se aproximá-los; em casa, para ativar ações específicas de acordo com preferências pessoais, como por exemplo ligar as luzes ao chegar, ou o ar-condicionado caso permaneça por algum tempo no mesmo ambiente; em viagem, para fazer o *check-in* eletrônico a partir do PDA, ou acessar a Internet; em negócios, para receber informações sobre produtos e serviços de forma transparente ao se aproximar de lojas ou escritórios.

Até oito dispositivos podem se conectar simultaneamente utilizando *Bluetooth* formando uma rede chamada de *Piconet*, ilustrada na Figura 2.2. Nesta situação, um dispositivo funciona como mestre e os demais como escravos. O dispositivo mestre pode enviar uma informação para todos os escravos simultaneamente, como um vídeo em tempo real. Caso haja vários dispositivos espalhados em uma área maior que o alcance do enlace de rádio, ou com mais de oito dispositivos, a tecnologia *Bluetooth* introduz o conceito de *Scatternet*, onde existem mais de uma *Piconet* que compartilham pelo menos um

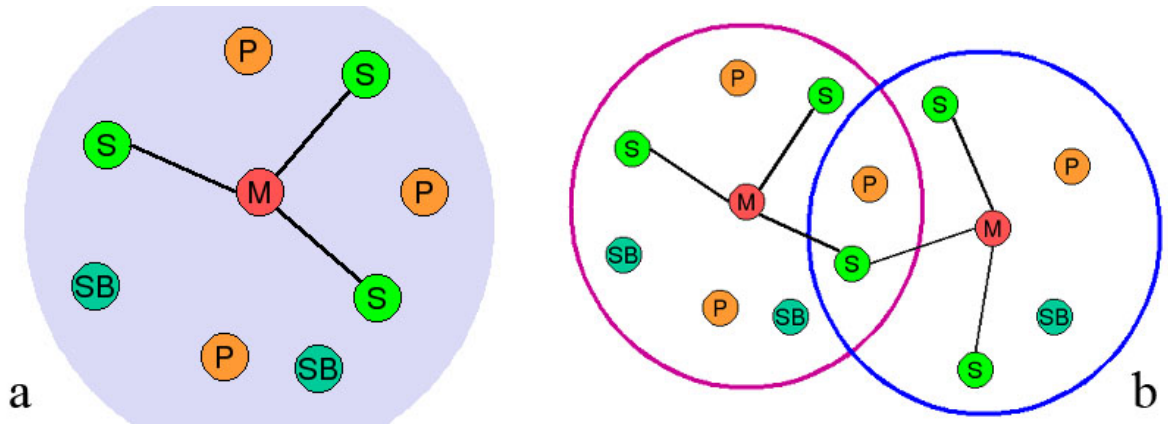


Figura 2.2: Diagrama das Piconet(a) e Scatternet(b)

dispositivo, responsável pelo roteamento das mensagens entre as *Piconets*.

O *Bluetooth* utiliza uma interface de rádio de curto alcance (10m), baixo consumo de energia, utilizando uma faixa de freqüência livre de licença (entre 2.4 GHz e 2.485 GHz ISM[1]), definindo 79 portadoras espaçadas de 1 MHz, atingindo uma taxa de transmissão básica em 1Mbps, capaz de transmitir voz e dados. A comunicação entre os dispositivos Bluetooth é feita através de um canal FH-CDMA *(Frequency Hopping - Code-Division Multiple Access)* que "salta" constantemente de freqüência para combater interferências e enfraquecimento do sinal. A cada segundo são realizados 1600 saltos de freqüência. Este mecanismo de *frequency hopping* auxilia na coexistência de dispositivos *Bluetooth* com outros (*non-hopping*) sistemas ISM que se encontram na mesma localização.

A especificação *Bluetooth* divide a pilha de protocolos em três grupos lógicos: o grupo de protocolos de transporte, o grupo de protocolos de midlleware e grupo de aplicação, como ilustrado na Figura 2.3.

O grupo de protocolos de transporte trata das questões relativas a localização de dispositivos e gerenciamento dos *links* físico e lógico com as camadas superiores. Neste contexto, não um relacionamento direto entre os protocolos de transporte da tecnologia *Bluetooth* com os protocolos da camada de transporte do modelo OSI (utilizado na especificação de protocolos de rede), e sim com às camadas OSI física e de enlace de dados. As camadas de Rádio Freqüência (RF), *Baseband*, *Link Manager* e *Logical Link Control and Adaptation* (L2CAP) estão incluídas no grupo de protocolos de transporte. Estes protocolos suportam tanto as comunicações síncronas quanto assíncronas e todas estas

[1] *Industrial, Scientific, Medical*

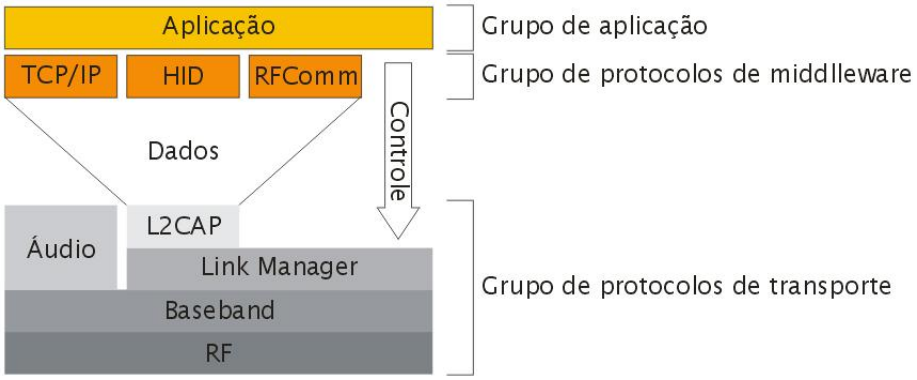


Figura 2.3: Pilha de protocolos *Bluetooth*

são indispensáveis para a comunicação entre dispositivos *Bluetooth*.

O grupo de protocolos de middleware inclui protocolos de terceiros e padrões industriais. Estes protocolos permitem que aplicações já existentes e novas aplicações operem sobre links *Bluetooth*. Protocolos de padrões industriais incluem *Point-to-Point Protocol* (PPP), *Internet Protocol* (IP), *Trasmission Control Protocol* (TCP), *Wireless Application Protocol* (WAP), etc. Outros protocolos desenvolvidos pelo próprio SIG[2] também foram incluídos neste grupo: RFComm, para o suporte às aplicações legadas e o *Service Discover Protocol* (SDP), que permite que dispositivos obtenham informações sobre serviços disponíveis em outros dispositivos.

O grupo de aplicação consiste das próprias aplicações que utilizam links *Bluetooth*. Estas podem incluir aplicações legadas ou aplicações orientadas especialmente para o uso do *Bluetooth*. Pode-se resumir as características das camadas da pilha de protocolos da seguinte forma:

- **Camada de Rádio (RF):** dedicada ao projeto dos transmissores de rádio;
- **Camara *Baseband*:** definie como dispositivos localizam outros dispositivos e como estabelecem conexões. Comporta a definição dos papeis mestre e escravo, estratégias de detecção de erros, criptografia, transmissão e retransmissão de pacotes. Esta camada suporta dois tipos de conexões: *Synchronous Connection-Oriented* (SCO, conexões prioritárias basicamente utilizadas na transmissão de voz) e *Assynchronous Connection-Less* (ACL, capaz de enviar pacotes de tamanho variável mas sem prioridade de transmissão).

[2] *Special Interest Group*

- ***Link Manager*:** implementa o *Link Manager Protocol* (LMP) responsável por gerenciar as prioridades de transmissão no meio entre os dispositivos.

- **Camada L2CAP:** *Logical Link Control and Adaptation*, serve de interface entre os protocolos de transporte e as camadas superiores. É responsável pela fragmentação e remontagem de pacotes.

Quanto ao processo de comunicação, um dispositivo *Bluetooth* pode estar em um dos seguintes estados: espera, solicitação, página, conectado, transmissão, bloqueado, escuta e estacionado (Figura 2.4). Um dispositivo está no estado de espera quando está ativo porém ainda não se juntou a uma *piconet*. Este entra no estado de solicitação quando envia requisições de busca de outros dispositivos com os quais possa se conectar. Quando o dispositivo em questão já é mestre de uma dada *piconet*, pode entrar no estado de página quando estar enviando mensagens à procura de dispositivos que possam se juntar a sua *piconet*.

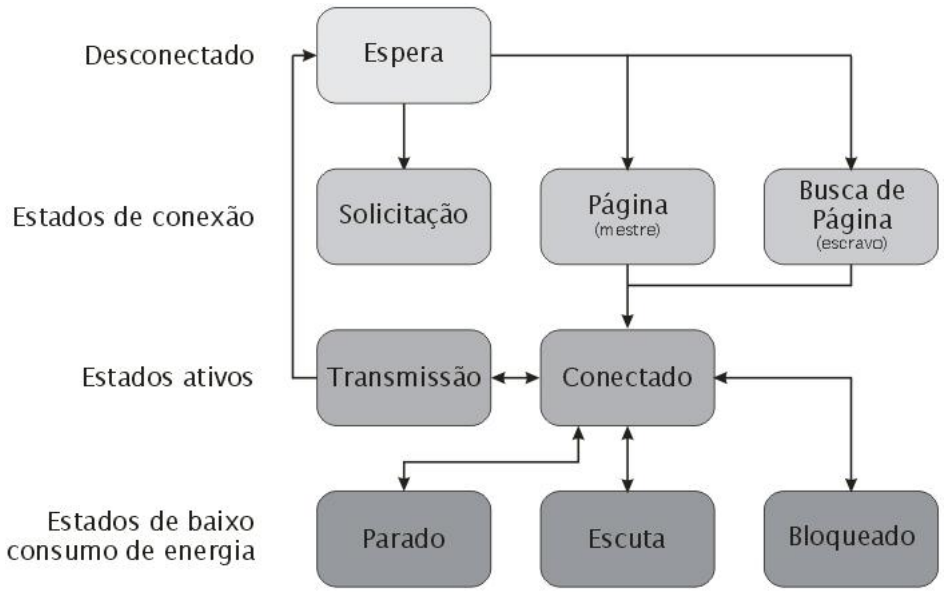


Figura 2.4: Diagrama de estados da especificação *Bluetooth*

Assim que uma comunicação é bem sucedida entre um mestre e seu novo escravo, este entra no estado conectado. Quando conectados, os escravos podem transmitir dados quando seu mestre permitir fazê-lo. Durante a suas transmissões, os escravos estão no estado de transmissão. Ao seu término, retornam ao estado conectado.

O estado de escuta é um estado de baixo consumo de energia onde o escravo "dorme" por um determinado espaço de tempo (determinado pelo mestre) até poder

transmitir novamente. De forma semelhante, no estado de bloqueio há um baixo consumo de energia e o escravo se torna inativo por um determinado tempo. Neste estado, não há transmissão de dados, porém o escravo ainda é considerado parte da *piconet* em questão. Caso um escravo não tenha dados a serem enviados por um longo período de tempo, este pode entrar no estado de estacionado. Neste estado, o escravo perde sua identificação na piconet cedendo seu endereço para outro dispositivo.

O *Bluetooth* é um tecnologia que aborda vários pontos chaves que facilitam sua vasta adoção: é uma especificação aberta e publicamente disponível; sua tecnologia sem fio de curto alcance permite que dispositivos periféricos se comuniquem facilmente, sem os problemas causados pelos cabos, conectores de diferentes formas, tamanhos e números de pinos; a especificação suporta a transferência tanto de voz quanto de dados, tornando-se ideal para a comunicação de dispositivos heterogêneos; e utiliza uma faixa de freqüências não regulamentada e amplamente disponível em qualquer lugar do mundo.

2.3 Desafios do Desenvolvimento de *Software*

A seção 2.1 apresentou características essenciais de ambientes de computação móvel. Estas características impuseram um conjunto de desafios que vem sendo trabalhados por pesquisadores da área. Esta seção apresenta os principais desafios relacionados ao desenvolvimento de *software* voltados aos ambientes móveis.

2.3.1 Heterogeneidade de Dispositivos

Dispositivos móveis são excelentes para tarefas rápidas e corriqueiras enquanto computadores de mesa provêem um conjunto de recursos que suportam interações mais complexas. Realizar a tarefa apropriada no dispositivo apropriado é um desafio que os analistas enfrentam ao desenhar aplicações que permitem acesso a partir de múltiplos dispositivos [IF03].

Prover interfaces e funcionalidades apropriadas para cada modelo [SS04], observando as diferenças entre os dispositivos, e adaptando as aplicações de acordo com o ambiente onde estão sendo executadas, provendo assim um alto grau de portabilidade, ainda hoje se apresenta como um dos grandes desafios provenientes da heterogeneidade

de dispositivos móveis. Desde o reconhecimento das características do visor até sua capacidade de processamento e conectividade, cada detalhe deve ser analisado para, por exemplo, decidir se uma computação deverá ser realizada no dispositivo móvel de forma autônoma ou na rede infra-estrutura, utilizando o nó móvel apenas para a exibição e coleta de informações.

2.3.2 Riscos de Segurança

Em uma rede sem fio o sinal é propagado pelo ar, onde qualquer receptor pode interceptá-lo. Esta peculiaridade das redes sem fio trás consigo alguns riscos [AGIS00]. Qualquer dispositivo no raio de alcance do enlace de rádio de uma rede sem fio pode interceptar o sinal e obter acesso a qualquer informação nele contido. A interceptação de sinais pode também revelar informações que proporcionam a utilização indevida dos serviços disponibilizados. Desta forma, questões de autenticação e criptografia se tornam fundamentais a fim de preservar a segurança das comunicações.

Os recentes avanços tecnológicos dotaram os telefones celulares, *Smartphones* e PDAs com memórias persistentes maiores e intercambiáveis, assim como processadores mais rápidos, dando oportunidade aos usuários de utilizarem tais dispositivos para armazenar (e processar) informações pessoais. Contudo, utilizar dispositivos móveis como repositórios de dados pessoais e a partir deles compartilhá-los levanta questões sobre a segurança da informação [LLZW04, ROPT05, RIS05, WPD+02]. Questões como quem pode acessar um dado recurso compartilhado, assim como o que acontecerá com tais recursos caso o dispositivo móvel seja roubado, devem ser analisadas a fim de prover soluções compatíveis com o perfil de uso dos aparelhos [KPB+04].

2.3.3 Restrições de Energia

É natural que o usuário deseje que seu dispositivo móvel esteja sempre disponível, uma vez que o mesmo armazena informações pessoais que podem ser requisitadas a qualquer instante. Porém, a capacidade das baterias não cresceu no mesmo passo que a capacidade de armazenamento e processamento, levantando questões sobre gerenciamento de energia antes deixadas em segundo plano [IF03].

Enlaces de rádio, visores maiores e tarefas que exijam grande processamento

(ex. compressão ou criptografia) demandam fluxos constantes de energia e podem consumir rapidamente a capacidade das baterias atuais, comprometendo sua utilização. Desenvolver aplicações que necessitem de pouca energia, utilizar programas específicos para o seu gerenciamento, assim como construir aplicações capazes de adaptar seu comportamento de acordo com o nível de energia disponível, constituem-se em um importante desafio no desenvolvimento de aplicações para dispositivos móveis.

2.3.4 Heterogeneidade de Rede

As diversas tecnologias de rede sem fio existentes possuem características diferentes como, por exemplo, a área de cobertura, endereçamento, largura de banda, latência, taxa de erros e etc. A visão da computação móvel implica em movimentar-se livremente com seu dispositivo sem interromper a computação e comunicação. Como exemplo deste comportamento podemos citar a exibição de um vídeo em tempo real. Variações na qualidade da conexão sem fio ocorrerão quase que inevitavelmente enquanto o usuário se desloca. Desta forma, esse processo de mobilidade envolve uma mudança de comportamento das aplicações no intuito de melhor adaptar-se às novas condições de conectividade. Esta adaptação do comportamento das aplicações gera questões que podem ser tratadas ao nível do sistema (de forma transparente à aplicação), através da própria aplicação (*laissez-faire* [AGIS00]) ou através de um conjunto sincronizado de ações tomadas pelo sistema, assim como pela aplicação.

Obviamente, colocar toda a responsabilidade pela adaptação nas aplicações gera uma grande carga de trabalho para os desenvolvedores, além de exigir um suporte apropriado dos sistemas de baixo nível. Por outro lado, a abordagem transparente pode não ser suficiente, uma vez que a aplicação não terá controle algum sobre a adaptação escolhida pelo sistema. Vejamos o exemplo citado anteriormente: o usuário utiliza seu dispositivo móvel para assistir um vídeo em tempo real transmitido por servidores na rede sem fio. Ao mover-se da área de boa conectividade para outra onde há menor largura de banda, um sistema transparente poderia simplesmente decidir que não há mais condições de exibir o vídeo, enquanto que uma aplicação adaptativa poderia solicitar a redução da resolução e/ou profundidade de cores do vídeo para manter sua exibição.

No tocante à adaptação da aplicação, também existe a possibilidade de se alterar o conjunto de funcionalidades disponíveis ao usuário, selecionando quais delas são

mais apropriadas às condições de rede existentes. Pode-se optar por adaptar os dados, quando por exemplo alteramos sua qualidade, como no caso do vídeo. Ou pode-se ainda utilizar uma abordagem mista, deixando a cargo do sistema selecionar a melhor tecnologia de rede disponível (quando existe mais de uma), e a cargo da aplicação reduzir a resolução do vídeo ao observar uma queda na qualidade da conexão. Desta forma, fica evidente o desafio presente nas questões de adaptabilidade perante a grande heterogeneidade de rede existente.

2.3.5 Desconexão e Fraca Conectividade

Mesmo com o grande número de torres de celular ou pontos de acesso, sempre haverá ponto escuros onde não há sinal e, portanto, não há conectividade. Neste cenário, usuários devem trabalhar de forma desconectada por intervalos de tempo preferencialmente curtos, para em seguida, quando houver conexão, sincronizar seus trabalhos [IF03].

Sistemas tradicionais dependem fortemente da rede e podem ficar bloqueados aguardando por respostas perdidas durante uma desconexão. A comunicação sem fio é muito susceptível à desconexões. Quanto mais autônomo um dispositivo móvel for, melhor será seu suporte à desconexão. O desafio, então, é tornar estes intervalos de desconexão o mais transparente possível, como por exemplo, transferindo funcionalidade e dados para o dispositivo móvel.

Ainda existem os problemas advindos da fraca conectividade. Baixa largura de banda, constantes perdas de pacotes e altas latências são problemas comumente enfrentados por usuário de dispositivos móveis [Sta02]. Técnicas de compressão, transmissão de dados em bloco para evitar várias transmissões curtas consecutivas e escrita retardada são algumas das técnicas freqüentemente empregadas para minimizar o impacto da fraca conectividade. O desafio é, portanto, determinar quais destas técnicas melhor se adapta aos dispositivos móveis levando em consideração seus recursos disponíveis e ajustando seu comportamento na medida em que estes recursos são consumidos.

2.4 Conclusão

Este capítulo apresentou fundamentos da computação móvel utilizados no con-

texto de nosso trabalho. Foram descritas as principais características de ambientes da computação móvel como a descentralização de serviços, a diversificação de dispositivos, o uso de interfaces simples e fáceis de usar e as variadas tecnologias de rede que permitem conectividade a qualquer hora e em qualquer lugar. Deu-se especial atenção à tecnologia *Bluetooth*, dado que ela foi utilizada como base para a implementação do framework SNU.

Apresentamos ainda importantes desafios enfrentados pelos desenvolvedores de *software* voltados à ambientes de computação móvel. Destacou-se as dificuldades inerentes à heterogeneidade da rede, aquelas relacionadas à enorme diversidade de características de *hardware* e *software* de dispositivos móveis, aspectos específicos de segurança em ambientes móveis, restrições de energia e a ocorrência de desconexões ou conectividade intermitente. Estes aspectos, aliado às peculiaridades das redes *ad-hoc* espontâneas, como a inexistência de uma infra-estrutura que pudesse abrigar serviços centralizados, foram levados em consideração no projeto do framework SNU.

3 Trabalhos Relacionados

Com o avanço contínuo das tecnologias de miniaturização e armazenamento de energia, o poder computacional dos dispositivos móveis vem crescendo rapidamente [SS04]. Ambientes de computação móvel, com sua capacidade de interferir nas interações sociais [Bea05, ROPT05], têm atraído cada vez mais atenção da comunidade científica. Existem atualmente diversos projetos relacionados à computação móvel, com diferentes objetivos, aspectos de implementação e modelos de aplicações suportadas. Este capítulo descreve relevantes projetos de computação móvel que possuem relação com o objetivo do trabalho apresentado nesta dissertação.

3.1 MIRES

Com o objetivo de tornar simples e eficiente o compartilhamento de dados em dispositivos móveis (basicamente telefone celulares) e levando em consideração sua capacidade reduzida de armazenamento, assim como seu pequeno poder computacional, o Departamento de Engenharia da Computação e Tecnologia da Informação juntamente com o Departamento de Ciência da Computação da Universidade de Hong Kong desenvolveram o MIRES, *Mobile Information Resource Exchange System* [LLZW04]. O projeto defende a utilização de um modelo cliente-servidor, semelhante aos bancos de dados distribuídos (DDB), para o compartilhamento de informações em dispositivos móveis utilizando redes infra-estruturadas e serviços centralizados.

Os dados compartilhados residem no servidor e são seletivamente enviados aos clientes na medida em que haja necessidade. A Figura 3.1 mostra como o MIRES está dividido em três componentes principais: um sistema global de gerenciamento (GMS), servidores de banco de dados de recursos móveis (MRDB) e um sistema informativo (*Bulletin Board*). Arquiteturalmente, o MIRES é um sistema totalmente distribuído baseado em um bancos de dados descentralizados (MRDB). A partir desta abordagem, apenas alguns dados ficam armazenados nos clientes móveis e a maior parte fica nos

servidores. Quando um usuário compartilha um dado, ele o envia ao servidor mais próximo de sua célula que o disponibiliza para os outros clientes, sendo assim, o MIRES funciona como um facilitador que gerencia o compartilhamento das informações.

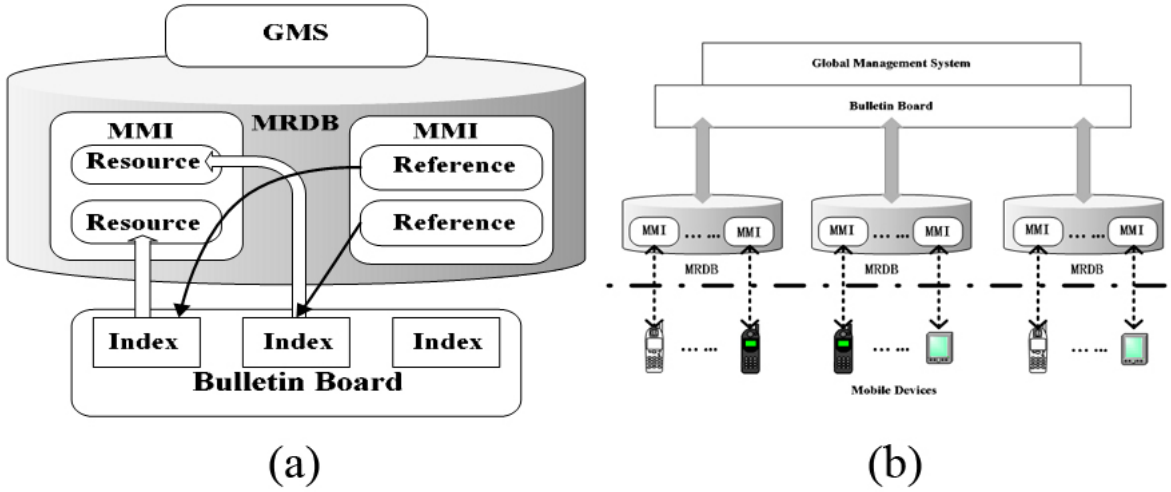


Figura 3.1: Arquitetura do MIRES

A transferência de dados entre servidores e clientes é feita utilizando SMS para o envio de comandos e recebimento de notificações e GPRS para transferência do recurso em si. Cada usuário possui uma imagem exata (MMI, *mobile mirror image*) do seu dispositivo móvel armazenada no servidor (Figura 3.1.b), identificada pelo número do telefone. O GMS só acessa as MMIs e os telefones só podem compartilhar seus recursos através das MMIs. Cada telefone só consegue acessar o conteúdo de sua própria MMI. As MMIs armazenam dados ou referências aos dados em outras MMIs e têm seu conteúdo indexado e disponibilizado no sistema informativo, que pode classificar e/ou agrupar o mesmo. Quando um cliente solicita um recurso, deverá localizá-lo no sistema informativo para em seguida solicitar sua transferência. Para minimizar o espaço utilizado pelo sistema quando um usuário solicita um recurso, ele terá uma nova entrada em sua MMI apontando para o novo recurso e não uma cópia do mesmo. Esta entrada só será criada caso o dono do recurso tenha lhe dado permissão de acesso. No momento da transferência, o MIRES segue as referência entre MMIs, localiza o recurso em si e o transfere para a memória local do cliente.

Alguns pontos em aberto são citados. As limitações da tecnologia Java em dispositivos móveis dificultam substancialmente o desenvolvimento de aplicativos complexos, uma vez que a JVM possui apenas algumas classes básicas e não há como acessar diretamente os recursos nativos do dispositivo. A JVM é executada em um ambiente

fechado e nem sempre os fabricantes implementam o padrão na íntegra, especialmente no que tange os pacotes adicionais conhecidos por JSR (*Java Specification Requests*). Um outro aspecto citado ocorre quando um recurso não está mais sendo referenciado. O sistema pode tratá-lo como lixo, o que pode levar a perda irreparável do mesmo. Contudo, o fato de não ser mais referenciado no presente não implica em sua inutilidade obrigatória. A realocação de dados entre MRDB para minimizar o tráfego na rede é outro ponto em questão. A autenticação deve ser contemplada, porém aumentará substancialmente o tráfego na rede e, conseqüentemente, o custo de utilização do sistema.

A filosofia de compartilhamento do MIRES é centrada nos dados, utilizando uma arquitetura infra-estruturada, baseada em redes de alta confiabilidade e com controle de acesso centralizado. Este projeto também não prevê suporte à informações de contexto. Estes são os principais aspectos que diferenciam o MIRES do framework proposto nesta dissertação. Por outro lado, a capacidade de produzir e compartilhar dados, dando suporte a um controle de acesso, a partir dos dispositivos móveis e o uso de J2ME são os principais pontos de confluência entre estes trabalhos.

3.2 PGWW

A cada nova geração de dispositivos móveis são introduzidos equipamentos (PDAs, celulares e etc.) cada vez mais capazes de gerar, armazenar, processar e compartilhar conteúdo multimídia. A crescente popularização destes equipamentos tem lhes dado um caráter mais pessoal, uma vez que seus usuários armazenam um número cada vez maior de dados pessoais diversos, como fotografias, vídeos, agenda de compromissos, contados e afazeres. Alguns dispositivos são ainda capazes de gerenciar informações de contexto, como sua posição geográfica, possibilitando a construção de aplicativos capazes de adaptar seu comportamento ao reconhecer mudanças em seu ambiente computacional. Com os objetivos de melhor utilizar tais recursos e prover um mecanismo simples e eficiente de compartilhamento do conteúdo digital, os laboratórios da DoCoMo USA desenvolveram o *Personalized Group Wide Web*, PGWW [RIS05].

O PGWW é uma ferramenta de compartilhamento de conteúdo que disponibiliza suas funcionalidades a partir do próprio dispositivo utilizando uma interface semelhante às encontradas em ambientes Web. O PGWW é baseado em um *middleware* de

serviços, que provê funcionalidades como segurança, localização de dispositivos e *caching* de requisições, intermediando o acesso ao conteúdo compartilhado e diminuindo as transferências de dados diretamente entre dispositivos.

Os conteúdos compartilhados nem sempre são enviados ao servidor para que não haja desperdício dos recursos nos clientes, principalmente com relação à energia. A infra-estrutura de serviço de armazenamento os transfere para o servidor apenas no momento da primeira requisição de acesso, armazenando-os e controlando suas versões, para que as futuras solicitações possam ser atendidas a partir da última versão disponível. Este modelo evita conexões desnecessárias com o dispositivo móvel provedor do recurso compartilhado.

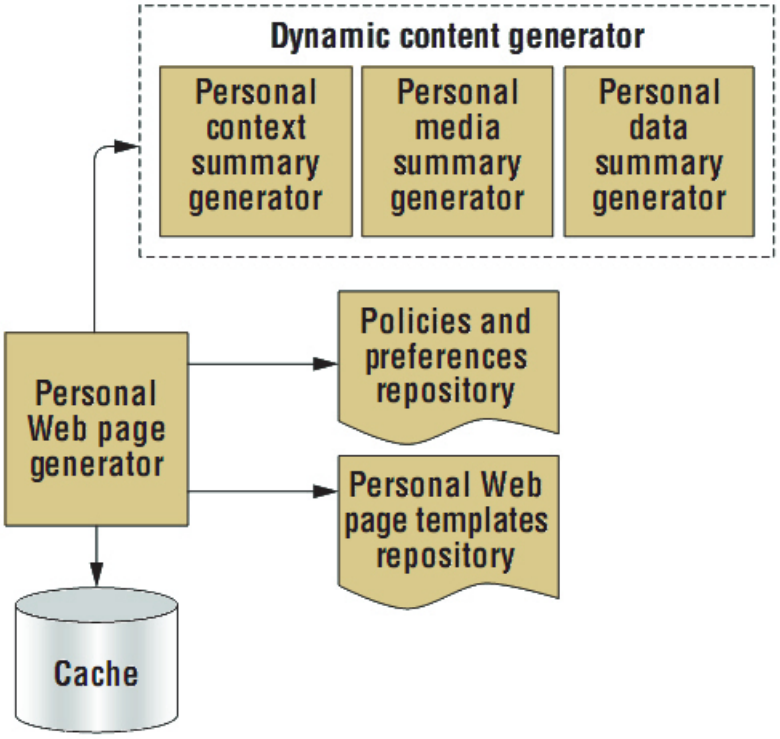


Figura 3.2: Arquitetura do PGWW

A Figura 3.2 mostra os vários elementos que compõem o *middleware* proposto: gerador de conteúdo dinâmico, repositório de preferências e políticas, repositório dos *templates* das páginas Web, *cache* e gerador da página Web pessoal. O gerador de conteúdo dinâmico é responsável por inspecionar os recursos compartilhados e criar um documento XML com todas as informações pertinentes ao compartilhamento. O repositório de preferências e políticas armazena as informações que controlam os aspectos de exibição da página Web contendo os recursos compartilhados assim como aspectos de segurança (vis-

ibilidade e controle de acesso). O repositório dos *templates* das páginas Web armazena as diversas personalizações possíveis da página Web de compartilhamento. O *cache* localiza, armazena e provê a reutilização total ou parcial dos recursos compartilhados. Por fim, o gerador da página Web pessoal utiliza as informações gerenciadas pelos outros serviços e cria a página Web combinando as diversas informações estáticas e dinâmicas da infra-estrutura.

O PGWW não pretende substituir a WWW e sim complementar a experiência de compartilhamento dos usuários. O atual modelo de compartilhamento de recursos do PGWW deverá evoluir pra um modelo mais sofisticado baseado em serviços como ocorreu com a WWW. Atualmente o protótipo desenvolvido foi escrito em J2ME e utiliza mecanismos simples para a construção da página Web.

O PGWW é um sistema construído sobre um *middleware* de serviços voltado ao compartilhamento dos dados gerados diretamente no dispositivo móvel, como fotos, sons e vídeos. Assim como o MIRES, o PGWW é baseado em uma arquitetura infra-estruturada com a utilização de uma rede de alta confiabilidade e controle de acesso centralizado. A capacidade de controlar todos os aspectos do compartilhamento diretamente do dispositivo móvel, assim como a utilização de J2ME são os pontos comuns entre este trabalho e o SNU. Ambos têm por objetivo potencializar o uso de dispositivos móveis fornecendo uma ferramenta capaz de compartilhar recursos de forma simples e transparente, sem a necessidade de conhecimento prévio do usuário de serviços do *middleware* existente por trás da aplicação.

3.3 ContextPhone

Telefones celulares estão quase sempre ligados e possuem uma relação íntima com seus usuários que costumam nele armazenar as mais diversas informações pessoais, assim como ajustá-los com aparência e toques personalizados. Através destes dispositivos pode-se ainda inferir diversas informações de contexto, como pistas sobre a situação atual de seus usuários (em reunião, almoçando...) e sua localização.

Para tornar tais propriedades úteis em nosso dia-a-dia, a Universidade de Helsinki e o Instituto de Tecnologia da Informação de Helsinki propõem o *ContextPhone* [ROPT05], uma plataforma de código aberto desenvolvida em C++ exclusivamente

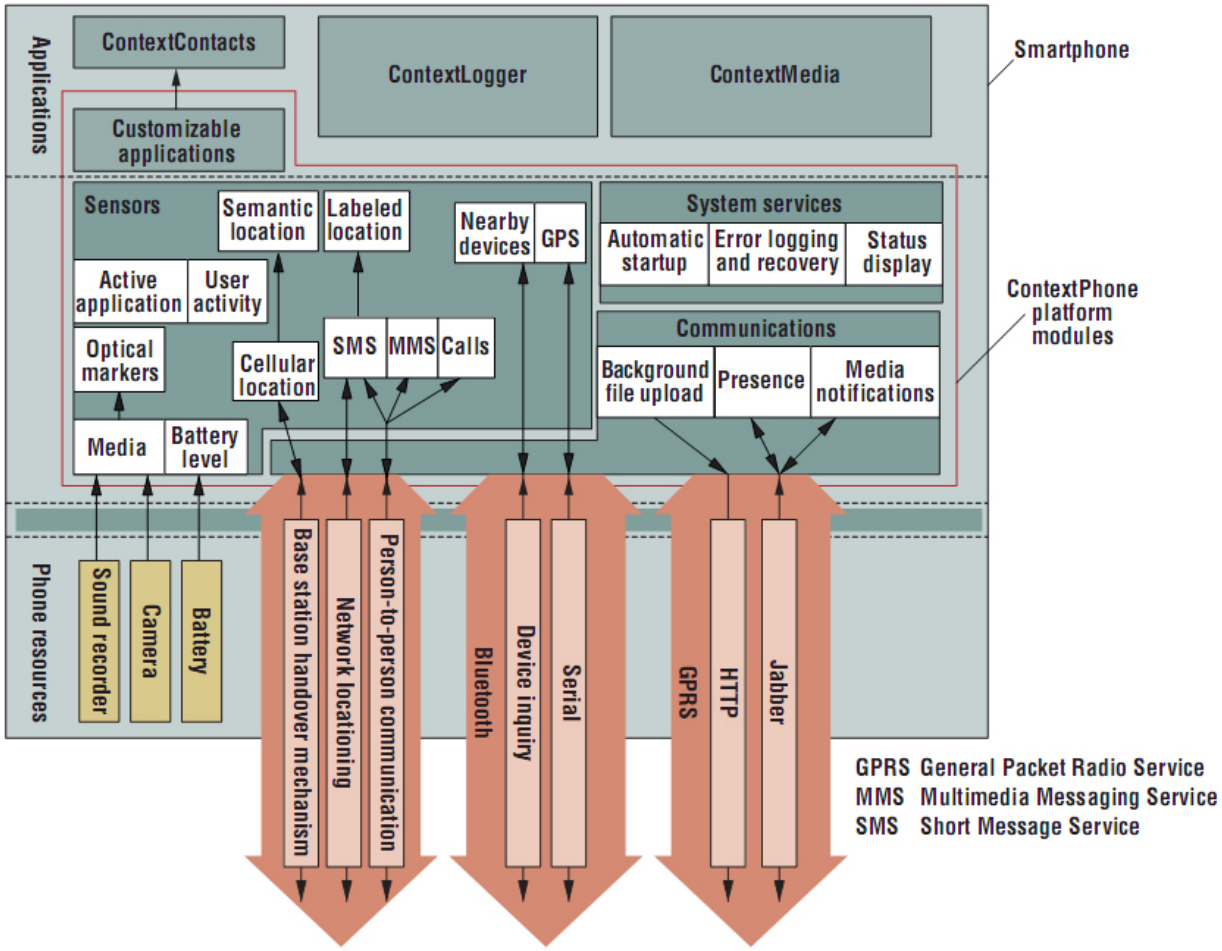


Figura 3.3: Arquitetura do ContextPhone

para os *Smartphones* que utilizam o sistema Symbian OS série 60. O *ContextPhone* possui os seguintes objetivos principais:

- Prover contexto como um recurso;
- Adicionar informações de contexto às aplicações existentes (especialmente agenda de contatos, mensagens e chamadas);
- Assegurar a robustez da solução a partir da persistência de toda informação relevante assim como do tratamento de erros sem necessariamente solicitar a intervenção do usuário;
- Permitir que o usuário controle a interpretação das informações de contexto, como por exemplo: se altero o perfil do celular para "silencioso", posso querer indicar que não desejo receber chamadas. Contudo, se apenas reduzo o volume do toque para o mínimo possível, estaria indicando apenas que não posso fazer barulho, mas posso perfeitamente responder uma mensagem;

- Disponibilizar componentes reutilizáveis e de fácil extensão que permitam um desenvolvimento rápido de aplicações consumidoras das informações de contexto.

A plataforma do *ContextPhone*, Figura 3.3, é dividida em quatro módulos:

1. Sensores, que coletam informações sobre o meio-ambiente, como dados de um GPS ou utilização de uma célula;
2. Comunicações, que se conecta com serviços externos, utilizando os protocolos padrão da Internet via GPRS, *Bluetooth*, SMS ou MMS;
3. Aplicações personalizáveis, como por exemplo o *ContextLogger*, *ContextContacts* e *ContextMedia*;
4. Serviços do sistema, responsáveis por iniciar automaticamente serviços em segundo plano, assim como armazenar e recuperar mensagens de erro (*log*).

O módulo de sensores é utilizado para coletar, processar, armazenar e transferir dados de contexto. As informações geradas podem disparar ações personalizadas do próprio telefone ou do mundo externo. A implementação atual possui quatro tipos de sensores: localização (através de células GSM e GPS), interação com o usuário (incluindo aplicações ativas, status ativos/inativo, status de recarga e captura de media), comportamentos de comunicação (chamadas e tentativas, gravação de chamadas, envio e recebimento de SMS e MMS), e o ambiente físico (rede e dispositivos *Bluetooth*, reconhecimento ótico através da câmera).

O módulo de comunicação permite estabelecer conexões locais, via infravermelho e *Bluetooth*, assim como conexões remotas via GSM/GPRS, e suporta o envio de arquivos através de chamadas HTTP *post*. Os serviços de SMS, MMS e mensagens *Jabber* são utilizados para enviar comandos e informações de presença.

O módulo de serviços de sistema inicia e monitora serviços em segundo plano, iniciando-os no *boot* e reiniciando-os em caso de falha. A arquitetura do sistema é baseada em eventos, utilizando a filosofia de subscrever e publicar eventos.

Todos os serviços permitem operações desconectadas, utilizando filas de comandos, como por exemplo, enfileirando as solicitações de transferência de arquivos

quando há falhas de conectividade. Tais serviços também armazenam as últimas informações de rede disponíveis, exportando-as na forma de contexto de conectividade (largura de banda atual, latência, taxa de erros e etc.). Como forma de contornar erros temporários, como memória insuficiente, estes serviços também refazem certas operações automaticamente.

Entre as aplicações disponibilizadas, ressalta-se o *ContextLogger* que armazena informações de contexto como a mobilidade. Isto possibilita que sejam realizados estudos soube interações sociais e relacionamento entre a disponibilidade do usuário e seu contexto atual. Por exemplo, o sistema pode observar que sempre que o usuário está em determinada localização geográfica o volume do toque de seu celular é ajustado para o mínimo ou até mesmo desligado (*mute*). Esse "aprendizado" faz com que o sistema possa ajustar o volume automaticamente apenas observando as informações de contexto de mobilidade.

O *ContextContacts* vincula informações de contexto com suas entradas na agenda de contatos. Ou seja, o sistema recupera as informações de contexto de cada contato presente na agenda e as disponibiliza juntamente com suas informações convencionais, como nome, telefone e email. Desta forma, ao navegar pela agenda de contatos, o usuário visualiza por exemplo a disponibilidade atual do contato (livre, ocupado, estudando e etc.), recuperada através de uma informação de contexto exportada.

O *ContextMedia* atrela informações de contexto (como a localização) aos recursos de mídia. Assim é possível anexar automaticamente a uma determinada foto, além da data e hora, informações como o local onde fora tirada. Por exemplo, através do contexto de localização do próprio usuário é possível inferir que a foto foi tirada no escritório do Dr. Paulo. Ou ainda, através do contexto de localização dos contatos da agenda, pode-se anexar uma lista com o nome dos presentes naquele instante. Todos os recursos de mídia podem ser compartilhados transferindo-os via HTTP/GPRS para servidores em redes infra-estruturadas com a utilização dos serviços de comunicação.

O *ContextPhone* e o SNU possuem estreito relacionamento no tocante ao compartilhamento de mídia e a informação de contexto. Apesar de modular e expansível, o *ContextPhone* é um pacote de soluções direcionado para uma determinada plataforma (Smartphones Symbian OS série 60), não sendo possível sua execução em outros ambientes. O SNU, por sua vez, possui como um de seus requisitos ser portável para um grande número de dispositivos.

O *ContextPhone* é baseado nas tecnologias de comunicação disponíveis nas redes de celular (como SMS, MMS e GPRS) e não permite sua execução sem essa infra-estrutura. O SNU é voltado para as redes *ad-hoc* espontâneas, podendo ser utilizado com qualquer tecnologia de rede que opere neste modo.

3.4 The Personal Server

O *Personal Server* [WPD+02], ou Servidor Pessoal, foi desenvolvido através de uma pesquisa patrocinada pela Intel. Consiste em um dispositivo portátil, sem interfaces com o usuário, capaz de armazenar e processar informações. Seu acesso é realizado através das interfaces de dispositivos ao seu redor. A idéia básica é criar um dispositivo portátil, leve e pequeno, que não possua interface com o usuário (visores, teclados, apontadores e etc.), porém que seja capaz de utilizar as interfaces dos equipamentos ao seu redor, através de um *link* de rádio de curto alcance. Utilizando os monitores, teclados e apontadores de microcomputadores ou quiosques, o *Personal Server* pode estender sua capacidade de armazenamento utilizando o espaço físico antes dedicado aos dispositivos de entrada e saída de dados. Contudo, ele sempre dependerá completamente do "empréstimo" das interfaces de dispositivos vizinhos para ser acessado.

A infra-estrutura do sistema provê duas funções básicas:

1. Descoberta e conexão de dispositivos móveis, utilizando redes *ad-hoc* de forma automática, ou seja, sem a necessidade de intervenção do usuário. O dispositivo móvel executa, em intervalos de tempo pré-determinados, varreduras *Bluetooth* a fim de localizar computadores próximos capazes de emprestar suas interfaces. Uma vez localizado um computador disponível ao redor, o sistema estabelece a conexão *Bluetooth* sem a necessidade de intervenção do usuário.

2. Suporte a computação através de um navegador Web convencional e do gerenciamento remoto do sistema. O projeto utiliza dois módulos, um executado no dispositivo móvel e o outro executado nos computadores que emprestam suas interfaces com o usuário (*host*). O dispositivo móvel possui serviços de navegação em páginas Web com suporte a manipulação do seu sistema de arquivos e gerenciamento remoto. O *host* provê o serviço de direcionamento dos dados recebidos pelos seus

dispositivos de entrada de dados para o dispositivo móvel, assim como a exibição dos dados enviados pela saída de dados padrão *stdout* deste.

A implementação do protótipo do dispositivo móvel utiliza um processador *StrongARM* executando Linux, DRAM 64MB, interface *Bluetooth* com suporte ao protocolo TCP/IP, em uma velocidade máxima de 723kbps, armazenamento através de memória *flash* de 32MB para inicialização do sistema e expansão através de cartões de memória *Compact Flash* (CF). O sistema é alimentado por uma bateria de *lithion-ion* de 920mAhr e tem o tamanho aproximado de uma fita cassete. Desta forma, o dispositivo possui boa capacidade para armazenamento de dados, baixo consumo de energia e razoável poder de processamento, suficientes para contemplar as necessidades de processamento e armazenamento de usuários convencionais.

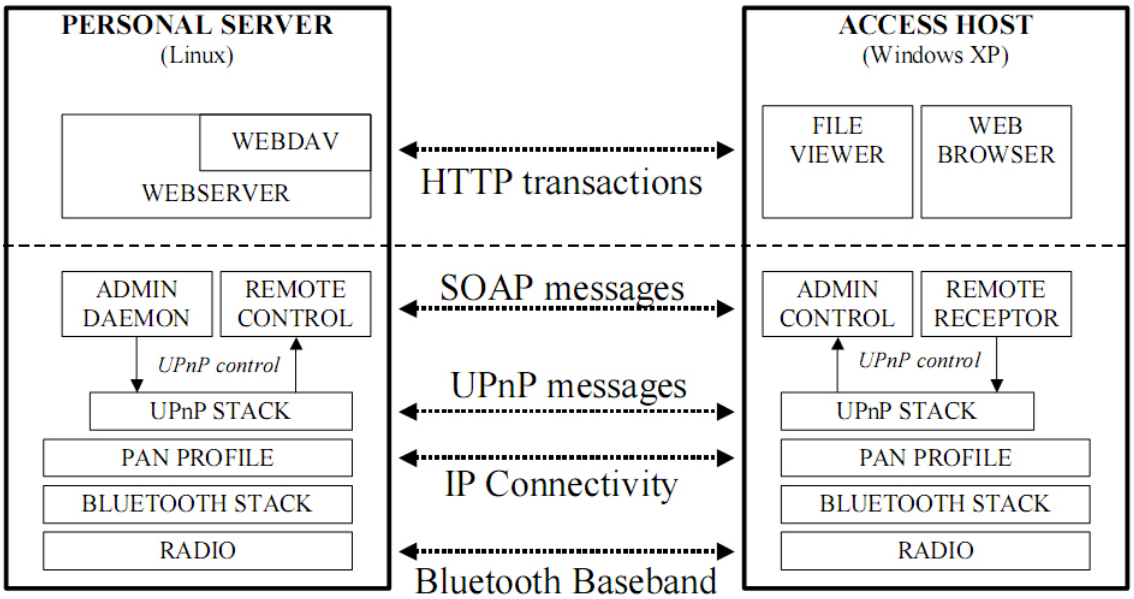


Figura 3.4: Arquitetura do PersonalServer

A Figura 3.4 mostra a arquitetura básica do sistema. No dispositivo móvel encontramos o suporte à conexão (*Bluetooth*), aos serviços Web através de um servidor Apache e as funções de administração e controle remoto (um serviço *deamon* personalizado). A administração remota permite a configuração do compartilhamento de dados, gerenciamento de usuários e senhas e informações básicas do dispositivo, como utilização da memória e bateria. O *host* deve executar Windows XP, com uma interface *Bluetooth* e o módulo do *middleware* responsável por localizar computadores próximos e prover protocolos de compartilhamento de seus dispositivos de entrada e saída de dados.

O foco em compartilhamento de recursos e acesso através de redes *ad-hoc* são os pontos de forte conexão entre o *Personal Server* e o framework proposto nesta dissertação. Contudo, o *Personal Server* é um dispositivo móvel sem interface com o usuário e que necessita de outros dispositivos para se comunicar. O SNU visa utilizar melhor os recursos de cada dispositivo móvel existente, provendo uma gama de serviços capazes de torná-los servidores pessoais de compartilhamento de dados, porém sem a necessidade de utilizar a interface de outros dispositivos.

3.5 Mobile Chedar

Chedar (*CHEap Distributed ARchitecture*) é um *middleware* que permite a localização e disponibilização de recursos ociosos em uma rede cabeada utilizando conexões ponto-a-ponto desenvolvido pela Universidade de Jyväskylä, Finlândia. É possível, por exemplo, localizar um nó que esteja ocioso e utilizar seus recursos livres para executar alguma tarefa de processamento computacional intensivo. Os nós mantém estatísticas de utilização e disponibilidade sobre arquivos, programas específicos e dispositivos de *hardware* como CPU, impressoras e memória de massa. O sistema foi desenvolvido em Java e, portanto, provê grande independência de plataforma.

Mobile Chedar [NMMJ05] é a extensão deste projeto para dispositivos móveis. Ele torna possível a utilização dos recursos compartilhados em uma rede *Chedar* a partir de dispositivos móveis capazes de executar J2ME. Desta forma, é possível registrar recursos livres da rede *Chedar* nos dispositivos móveis, tornando-os clientes destes recursos ou executar pesquisas sobre a rede.

Toda a comunicação móvel é feita com *Bluetooth*, uma vez que esta tecnologia se encontra amplamente disponível nos dispositivos móveis atuais. Para que um dispositivo móvel executando o *Mobile Chedar* possa acessar a rede cabeada ponto-a-ponto do *Chedar* é necessário que haja pelo menos um nó pertencente a rede *Chedar* que possua um *link* de rádio *Bluetooth* além da tecnologia de rede utilizada pela rede cabeada em si (em geral *Ethernet* - vide Figura 3.5). Este nó, que possue interface para as duas tecnologias de rede, funcionará como conexão da rede *Bluetooth* com os compartilhamentos existentes na rede *Chedar*. Desta forma, todas as mensagens dos serviços acessados pelos dispositivos móveis devem obrigatoriamente passar por ele.

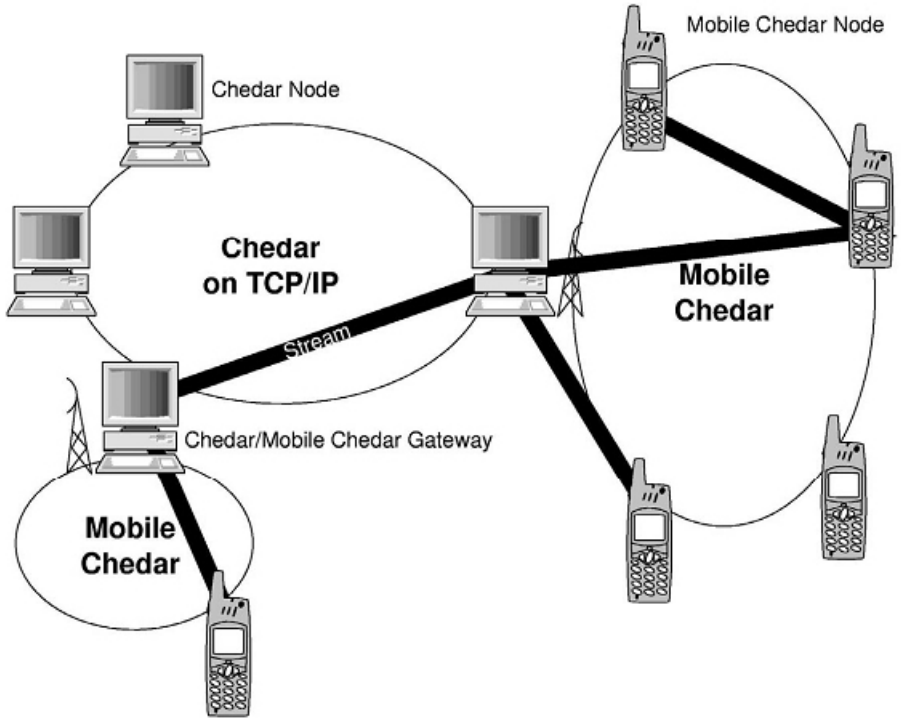


Figura 3.5: Visão Geral do Mobile Chedar

Foi implementada uma aplicação para exemplificar o uso da tecnologia proposta. A aplicação permite que alunos publiquem notas de aula e as compartilhem através da rede *Chedar*. As notas de aula podem ser alimentadas ou acessadas por qualquer nó da rede cabeada ou móvel, contudo a persistência é sempre provida pela rede *Chedar*. Todo conteúdo gerado nos nós móveis é sempre enviado e armazenado na rede *Chedar* cabeada.

A filosofia de compartilhamento deste projeto difere do SNU no momento em que parte do compartilhamento de recursos é realizado em redes infra-estruturadas. Não há compartilhamento de recursos dos próprios dispositivos móveis, que, por sua vez, sempre agem como clientes dos recursos da rede infra-estruturada. Contudo, ambos os projetos vislumbram soluções ponto-a-ponto de compartilhamento de informações e se preocupam com a portabilidade da solução, quando adotam o Java como ferramenta de desenvolvimento.

3.6 Outros trabalhos relacionados

Esta secção apresenta alguns projetos de computação móvel estudados que de alguma forma influenciaram o desenvolvimento do SNU apesar de apresentarem poucos

pontos em comum com o mesmo. Após a descrição de cada trabalho é realizada uma análise comparativa com a arquitetura proposta nesta dissertação.

3.6.1 *Collaborative Backup for Dependable Mobile Applications*

Marc-Olivier et. al. [KPB+04] propõem um *middleware* para o gerenciamento de cópias de segurança baseado na colaboração entre dispositivos móveis auto-organizados em redes espontâneas (MANETs). O objetivo é criar um serviço automático de backup e restauração de dados baseados na cooperação entre dispositivos móveis sem que haja nenhuma relação de confiança previamente estabelecida. Este serviço visa garantir a disponibilidade contínua de dados críticos gerenciados por dispositivos móveis comumente sujeitos a falhas de energia, danos físicos, perda ou roubo.

A idéia básica é permitir que um dispositivo móvel possa explorar dispositivos acessíveis em sua vizinhança para gerenciar cópias de backup de seus próprios dados críticos. São abordadas questões de segurança, cooperação, tolerância à falhas e confidencialidade. No tocante a natureza das redes espontâneas, esta cooperação se torna particularmente desafiante pelo intrínseco assincronismo (comunicações não confiáveis, particionamento, mobilidade, etc.) e pela ausência de conectividade contínua a um recurso global, como servidores de certificação e/ou autenticação, dispositivos de armazenamento estáveis e relógios globais.

A proposta de Mar-Oliver et. al. introduz uma forma interessante de se prover serviços de redundância a partir dos próprios dispositivos móveis, o que pode ser considerado em versões futuras do SNU.

3.6.2 *Replets*

O uso de aplicações Web em dispositivos móveis requer o tratamento a questões como desconexões ou fraca conectividade [Yua02]. No intuito auxiliar na resolução destes problemas, os laboratórios da DoCoMo USA apresentaram os *Replets* [ZII04], réplicas de serviços Web armazenadas diretamente nos dispositivos móveis, visando prover disponibilidade dos mesmos em períodos de desconectividade.

Segundo os autores, uma aplicação Web típica pode ser dividida em três partes:

a lógica relacionada a aplicação Web, a lógica da regra de negócio e os dados da aplicação. Para que haja replicação do serviço Web no dispositivo móvel, o mesmo deve ser planejado de forma a separar os serviços capazes de serem executados no dispositivo móvel daqueles executados na infra-estrutura, assim como deve ser determinado quais dados devem ser provisionados para que todo o sistema tenha o comportamento esperado. O serviço executado no dispositivo móvel será responsável por determinar se as respostas às solicitações do usuário serão processadas pela réplica do serviço no dispositivo ou pelo serviço original na rede. Desta forma, os *Replets* apresentam uma solução para o suporte ao trabalho desconectado, uma vez em que parte dos serviços e seus dados residem no próprio ambiente móvel.

A filosofia de levar os serviços ao dispositivo móvel e compartilhá-los a partir de lá são os pontos de confluência entre os *Replets* e o SNU. Contudo, o SNU parte do princípio de que não há uma infra-estrutura disponível e trabalha provendo as funcionalidades básicas de compartilhamento de recursos a partir do próprio dispositivo móvel.

3.6.3 *MoCA*

MoCA [SER+04] é um *middleware*, desenvolvido na PUC-Rio, para o desenvolvimento de aplicações sensíveis ao contexto em ambientes de computação móvel colaborativa. O objetivo do *middleware* é prover um ambiente flexível e extensível baseado em serviços para o desenvolvimento de aplicações para redes infra-estruturadas de computação móvel. A versão atual foi implementada para as redes sem fio 802.11, porém pode ser facilmente adaptado as redes de telefonia celular GPRS *(General Packet Radio Service)*.

As aplicações baseadas no *MoCA* devem ser desenvolvidas utilizando a API (*Application Programming Interface*) cliente chamada de `ClientAPI`. Se a aplicação for do tipo cliente/servidor, o servidor, geralmente executado na rede cabeada, deverá utilizar a API servidor chamada de `ServerAPI`. Estas API's escondem vários detalhes de utilização dos serviços básicos do *MoCA* e podem ser acessadas através de *proxies* próprios. `ClientAPI`, geralmente executada no dispositivo móvel, também é responsável por iniciar automaticamente o `MoCA Monitor`, um *deamon* responsável por coletar, em intervalos de tempo predefinidos, informações sobre o ambiente e o estado de execução do dispositivo, ou seja, a qualidade do sinal de rádio de todos os pontos de acesso visíveis, a utilização

da CPU, memória. bateria e etc.

Após a coleta dessas informações, o `MoCA Monitor` as envia para o *Context Information Service* (CIS), um serviço responsável pelo armazenamento e publicação de todas as informações de contexto dos dispositivos móveis. O CIS também recebe requisições de notificação de mudança de contexto feita por aplicações clientes ou servidores. Quando uma aplicação deseja ser notificada da mudança de um valor de contexto de um determinado dispositivo, envia um comando de subscrição para o CIS. Desta forma, após sofrer atualizações, o CIS as processa, verificando a ocorrência de eventos de interesse da aplicação, para em seguida notificá-las. Tanto a interface `ClientAPI` quanto a `ServerAPI` possuem métodos de acesso às informações gerenciadas pelo CIS.

Outros três serviços são disponibilizados pelo *MoCA*, o *Configuration Service* (CS), o *Location Inference Service* (LIS) e o *Discovery Service* (DS). O CS é responsável por armazenar e publicar a configuração dos dispositivos móveis disponíveis. Desta forma, as aplicações podem acessar o CS para conhecer os dispositivos móveis cadastrados e utilizar o CIS para solicitar notificações de alterações em seus contextos. O LIS é responsável por inferir a localização dos dispositivos a partir dos dados referentes aos pontos de acesso visíveis por cada um, coletados pelo CIS. O LIS mantém uma lista com a localização de cada ponto de acesso e assim pode inferir a posição dos dispositivos a partir da triangulação dos sinais de rádio visíveis por eles. O DS é responsável por armazenar as informações (nome, propriedades, endereço e etc.) de todos os serviços e aplicações disponíveis, registrados pelos métodos presentes na `ServerAPI`.

Tanto o *MoCA* quanto o SNU visam prover um arcabouço que facilite o desenvolvimento de aplicações colaborativas em ambientes de computação móvel. No entanto, o *MoCA* é baseado em redes infra-estruturadas enquanto o SNU baseia-se em redes *ad-hoc* espontâneas sem infra-estrutura. Desta forma, a arquitetura e soluções adotadas em cada projeto são fundamentalmente diferentes. Em cenários onde um espaço físico é apenas parcialmente coberto por uma rede infra-estruturada, estas duas abordagens podem ser complementares.

3.7 Resumo comparativo

Esta secção apresenta uma análise comparativa dos principais trabalhos apre-

sentados em relação ao SNU. Tomamos como base os seguintes critérios de comparação: tipo da solução (*framework*, que permite sua extensão, ou fechada), arquitetura (ponto-a-ponto ou cliente-servidor), tipo de rede (*ad-hoc* ou infra-estruturada), tecnologias de rede (*Bluetooth*, Wi-Fi, SMS, MMS ou GPRS), dispositivos alvo (celulares, PDA's, *smartphones* e outros), recursos compartilhados (dados e/ou serviços), local de execução dos serviços e de armazenamento dos dados (dispositivo e/ou servidores) e plataforma de desenvolvimento.

Podemos observar na Tabela 3.1 que com exceção do *Personal Server* todos os projetos relacionados são baseados em redes infra-estruturadas, não permitindo operação em modo *ad-hoc*, foco do projeto SNU. O *Mobile Chedar* trabalha em modo *ad-hoc* apenas no tocante à conexão do dispositivo móvel à rede cabeada através de um dispositivo mestre que possua acesso à ambas as redes. Contudo, os serviços são sempre providos pela rede infra-estruturada. Dessa forma, verificamos a necessidade de soluções em redes *ad-hoc* espontâneas, onde a disponibilidade de qualquer dispositivo é completamente imprevisível.

É interessante observar que todos os projetos se concentram no compartilhamento de dados, e, além do SNU, apenas o *Context Phone* provê suporte as informações de contexto. Contudo, o serviço de contexto do SNU se limita as informações de contexto da aplicação, enquanto que o *Context Phone* disponibiliza informações de contextos diversos, como localização, parâmetros de rede e informações próprias de aparelhos celulares, como o perfil em uso, volume do toque, recebimento de mensagens entre outras.

Quase todos os projetos foram desenvolvidos em J2ME, numa clara demonstração de preocupação com a portabilidade. Contudo, ao contrário do que ocorre com o SNU, cada projeto contempla uma arquitetura em particular, como *Smartphones* ou PDAs, pois necessitam de tecnologias de rede específicas como SMS/MMS, GPRS ou até mesmo o Wi-Fi, presente em alguns PDA's porém raramente presente em celulares ou *Smartphones*. Apenas o SNU, o *Context Phone* e o *Personal Server* utilizam *Bluetooth*, amplamente disponível nos dispositivos móveis citados. Dos trabalhos aqui relacionados, o SNU foi o único direcionado a qualquer dispositivo com suporte a J2ME (disponibilizando as JSR 75 e JSR 82 [II04,Inc05]), tornando viável sua utilização nos mais diversos dispositivos.

Projeto	**SNU**	**MIRES**	**PGWW**	**Context Phone**	**Personal Server**	**Mobile Chedar**
Solução	Framework	Fechada[a]	Framework	Fechada	Fechada	Framework
Arquitetura	Ponto-a-ponto	Cliente-servidor	Cliente-servidor	Ponto-a-ponto	Ponto-a-ponto	Cliente-servidor
Tipo de rede	Ad-hoc espontânea	Infra-estruturada	Infra-estruturada	Infra-estruturada	Ad-hoc	Ad-hoc acessando nós infra-estruturados
Tecnologia de rede	Bluetooth e outras	SMS / GPRS	Wi-Fi	SMS / MMS / GPRS	Bluetooth	Wi-Fi
Dispositivos	Qualquer c/ J2ME	Celulares c/ J2ME	PDA's c/ J2ME	Smartphone Symbian	Próprio (Linux)	PDA's c/ J2ME
Compartilhamento	Dados e serviços	Dados	Dados	Dados e contexto	Dados e serviços	Dados
Local dos serviços	Dispositivo	Servidores	Servidores	Dispositivo	Dispositivo	Servidores
Local dos dados	Dispositivo	Dispositivos e servidores	Dispositivos e servidores (cache)	Dispositivo	Dispositivo	Servidores
Plataforma	J2ME	J2ME	J2ME	Symbian OS 60 / C++	C++	J2ME

Tabela 3.1: Resumo comparativo dos principais trabalhos relacionados

[a]Solução completa para o problema estudado. Não visa o desenvolvimento de aplicações que utilizem recursos da solução proposta.

4 O *Framework* SNU

Neste capítulo analisamos a arquitetura do SNU, sua divisão em camadas com seus serviços, destacando seus detalhes, interfaces e interações. Organizamos nossa análise a partir da visão geral em camadas, detalhando suas funções, interfaces e serviços, a fim de demonstrar seu comportamento e suas funcionalidades perante as demais. Sempre que necessário destacamos as principais peculiaridades e dificuldades de implementação encontradas no decorrer do projeto, assim como discorremos sobre os motivos pelos quais optamos pela solução adotada.

4.1 Arquitetura do SNU

Para que possa ser flexível, robusto e possuir uma interface bem definida entre o *framework* e as aplicações, nós dividimos o SNU em camadas, ilustrado através da Figura 4.1, onde cada uma provê uma série de serviços para as demais. A divisão em camadas nos dá a oportunidade de adicionar ou substituir funcionalidades através da adição ou da troca de classes, desde que elas possuam o mesmo conjunto de interfaces.

Começando pelos fundamentos de comunicação, a Camada de Rede (*Network Layer*, a primeira camada de baixo para cima) é diretamente vinculada à tecnologia de rede utilizada e provê funções de entrada e saída de dados, encapsuladas em métodos que enviam comandos ao *framework* e recebem seus resultados. A Camada de Serviços (*Services Layer*, logo acima) comporta todos os serviços de usuário disponíveis em nossa solução. Ela possui todas as interfaces utilizadas pelos desenvolvedores durante a criação das aplicações do usuário (última camada sobre as demais). A comunicação sempre se dá entre camadas próximas, evitando assim que a aplicação do usuário se preocupe com os detalhes de implementação da tecnologia de rede em uso.

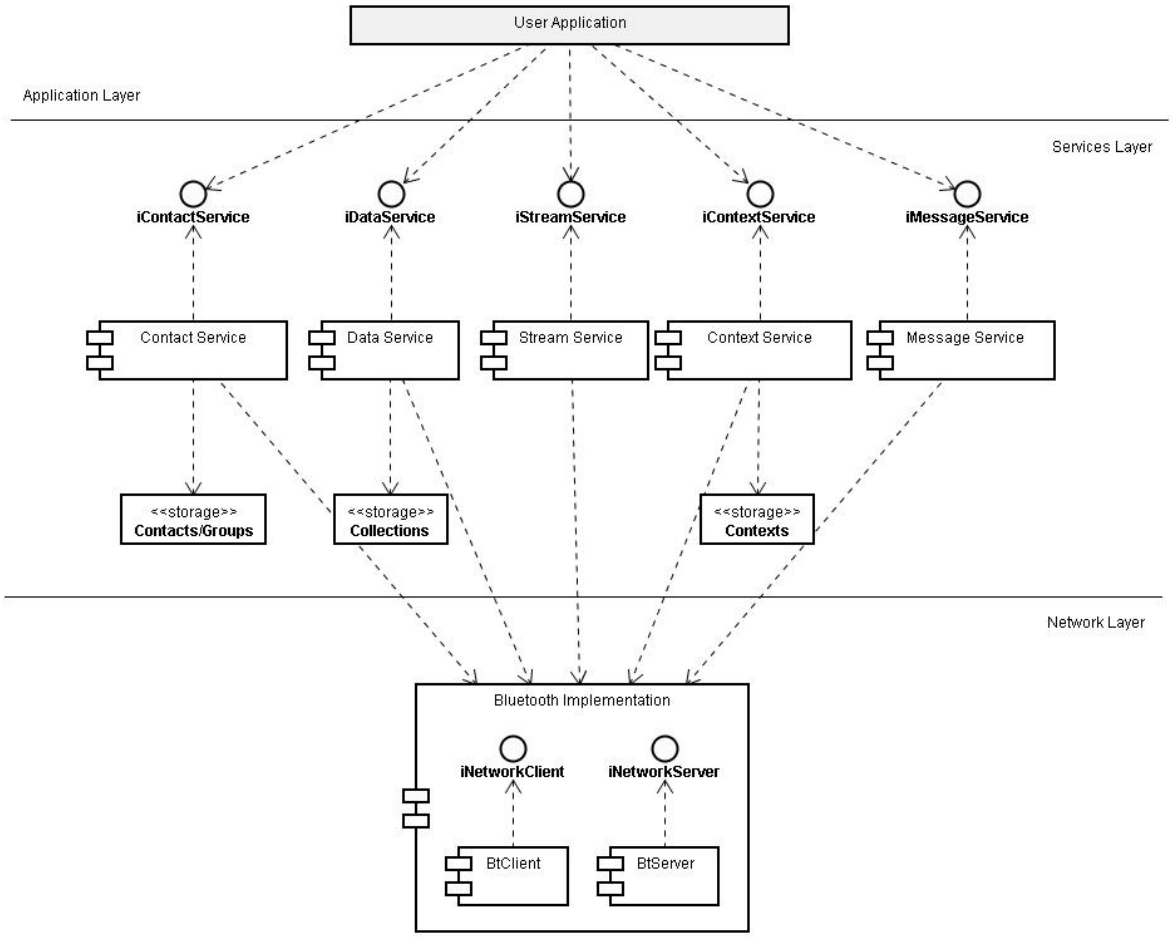


Figura 4.1: Diagrama dos componentes do SNU

4.2 A Camada de Rede

Na camada de rede são definidas as funcionalidades cliente e servidor, já que no modelo de rede *ad-hoc* os dispositivos devem sempre agir como ambos durante a execução das aplicações. Como forma de obter a independência de rede, o *framework* utiliza o Padrão Fábrica (*Factory Patern*) [GHJV94], provendo métodos para criação de objetos cliente e servidor de rede.

A Camada de Serviços não conhece nenhum detalhe de como ocorre a transferência de dados realizada pela Camada de Rede. Não há métodos de baixo nível, como enviar e receber *bytes* através de um *stream*, disponíveis para a Camada de Serviços. A Camada de Rede é responsável por enviar comandos de alto nível e aguardar pela sua confirmação de recebimento. Desta forma, a interface do cliente de rede possui um método para enviar apenas comandos de alto nível para um dispositivo alvo utilizando suas informações de conexão. Procurar por grupos de contatos compartilhados, requisi-

tar informações compartilhadas e solicitar a transferência de arquivos são exemplos de comandos de alto nível enviados pela Camada de Serviços através do cliente de rede. Do outro lado, a interface do servidor de rede possui métodos para escutar novas conexões, aceitá-las, receber e processar comandos e enviar suas respostas de volta ao dispositivo solicitante. Este modelo encapsula todo trabalho relacionado à rede e apresenta apenas simples interfaces para registrar, enviar e processar comandos nos dispositivos.

Cenários P2P sobre redes *ad-hoc* nos forçam lidar com o fato de que, a qualquer instante, um dispositivo previamente acessível pode estar fora de alcance sem qualquer aviso prévio. Logo, desenvolver rotinas de comunicação baseadas em conexões sempre ativas se torna impraticável [EH00]. Sendo assim, é necessário enviar todos os dados em uma única rajada (com confirmação de recebimento), pois não há garantias de que poderá existir uma segunda. Outro ponto que reforça o uso desta técnica de envio "em uma única rajada" é que as conexões de rádio em geral consomem muita energia. Desta forma, manter uma conexão ativa pode consumir a bateria do dispositivo tão rapidamente que tornaria a solução inviável [IF03].

Um outro aspecto importante levado em consideração foi a segurança. A Camada de Rede implementa o algoritmo de criptografia XXTEA (1998), a terceira versão do TEA - *Tiny Encryption Algorithm* [And03], uma solução de criptografia simétrica com uma chave fixa e conhecida por todos os dispositivos em operação. Este algoritmo de criptografia pode ser habilitado ou desabilitado pela aplicação através de parâmetros de configuração do *framework* durante sua inicialização.

```
1  package snu;
2
3  public interface iNetworkFactory {
4
5    public iNetworkServer createNetworkServer();
6
7    public iNetworkClient createNetworkClient();
8
9  }
```

Listagem 4.1: Interface iNetworkFactory

A interface `iNetworkFactory` (Listagem 4.1) segue o padrão de fábrica de objetos e define os métodos `createNetworkClient()` e `createNetworkServer()` responsáveis pela criação dos objetos de rede cliente e servidor, respectivamente. Caso haja mais de uma tecnologia de rede disponível pode-se, ao iniciar o *framework*, determinar qual objeto de rede será instanciado a partir de arquivos de configuração, linha de comando ou através

de algoritmos que verifiquem o ambiente em questão e determinem qual tecnologia utilizar. Atualmente a camada de rede está implementada para a tecnologia *Bluetooth* [Met99]. Porém, podemos facilmente estender este modelo para utilizar qualquer tecnologia de rede ad-hoc, bastando, para tanto, implementar as interfaces previamente definidas. A seguir detalhamos as interfaces cliente e servidor de rede, apresentando suas particularidades e detalhes de implementação.

4.2.1 Servidor de rede

A Listagem 4.2 exibe a interface `iNetworkServer`, onde existem apenas dois métodos: `startServer` e `register`. `StartServer` é responsável por iniciar o serviço servidor de rede recebendo como parâmetro os atributos de serviço *Bluetooth* que se fizerem necessário. Estes atributos são os identificadores de serviços da camada de aplicação disponibilizado via BSDP - *Bluetooth Service Discovery Protocol* [Met99, Gry]. A priori, o *framework* registra um único serviço BSDP que representa o SNU em si. Contudo, se a aplicação disponibilizar outros serviços, poderá registrá-los através do parâmetro de `startServer`, o que os tornará visíveis aos dispositivos vizinhos.

O segundo método disponível é o `register`. Ele é responsável por vincular um comando de alto nível pertencente a Camada de Serviços com um objeto serviço do *framework* capaz de processá-lo. Cada objeto serviço do *framework* utiliza este método em sua inicialização para registrar seus comandos. Desta forma, quando o objeto servidor de rede receber um comando saberá para qual objeto serviço deverá direcionar a chamada.

```
1  package snu;
2
3  import java.util.Vector;
4
5  public interface iNetworkServer {
6
7    void startServer(Vector attrib);
8
9    void register(String cmd, iService srv);
10
11 }
```

Listagem 4.2: Interface iNetworkServer

Uma classe que defina um serviço do *framework* deve necessariamente implementar a interface `iService`, Listagem 4.3, disponibilizando outros dois métodos: `netAnswer` para processar uma resposta recebida por um cliente após uma chamada a um servidor e `processCommand` para processar esta chamada no servidor. Esse modelo

nos permite estender a Camada de Serviços adicionando novos objetos serviços simplesmente implementando `iService`, provendo total desacoplamento entre as camadas em questão, não sendo necessária qualquer alteração nos objetos da Camada de Rede

```
1  package snu;
2
3  public interface iService {
4
5    void netAnswer(String command, String parameter,
6      String answer);
7
8    String processCommand(String cmdToProcess);
9
10 }
```

Listagem 4.3: Interface iService

4.2.2 Cliente de rede

A interface `iNetworkClient`, Listagem 4.4 armazena informações intrínsecas à rede, como o identificador de rede único do próprio dispositivo em uso, os dispositivos ao redor, seus nomes e identificadores. Para distinguir entres dispositivos em uso e assim poder reconhecê-los, o SNU armazena um identificador de rede único para cada dispositivo em questão, como o endereço MAC (apropriado nas implementações com 802.11) [Ass03] ou o endereço *Bluetooth* local (utilizado na atual implementação) [Met99].

```
1  package snu;
2
3  import java.util.Vector;
4
5  public interface iNetworkClient {
6
7    void startClient(iService callserv, String cmd,
8                     String parameter, String pconnStr);
9
10   Vector nearby_devices();
11
12   device findDevice(String id);
13
14   int findDeviceArrayPos(String id);
15
16   void scan_nearby_devices();
17
18   String device_id();
19
20   void set_outputFileName(String name);
21
22 }
```

Listagem 4.4: Interface iNetworkClient

Quando um serviço do *framework* deseja enviar um comando pela camada de rede, é feita uma chamada ao método `startClient` do objeto cliente de rede. Ao receber esta chamada, o objeto cliente de rede inicia uma *thread* que executará em paralelo,

providenciando a conexão com o dispositivo remoto e o envio do comando solicitado assim como seus parâmetros (caso existam). A utilização de uma *thread* em paralelo possibilita o uso assíncrono dos serviços, evitando esperas longas ou possíveis travamentos inconvenientes. Ao finalizar o envio do comando ao dispositivo remoto, a *thread* do objeto cliente de rede finaliza seu processamento. A resposta do comando enviado é recebida pelo objeto servidor de rede que está sempre ativo e escutando o canal de comunicação

Na interface `iNetworkClient` constam métodos para explicitamente localizar os dispositivos próximos (`scan_nearby_devices()`), encontrar um dispositivo pelo seu identificador de rede (`findDevice()`), retornar um vetor com os dispositivos ao redor (`nearby_devices()`), retornar a posição no vetor de dispositivos ao redor de um determinado dispositivo a partir de seu identificador de rede (`findDeviceArrayPos()`), retornar o identificador de rede do próprio dispositivo em questão (`device_id()`) e indicar qual será o nome local do próximo arquivo a ser recebido pelo *framework* (`set_outputFileName()`). Este último método é chamado pelo serviço que solicitou a transferência do arquivo remoto para indicar o nome completo do arquivo a ser criado localmente com o conteúdo transferido.

Implementação

O diagrama de classes da atual implementação em *Bluetooth* da Camada de Rede, Figura 4.2, mostra o relacionamento entre as entidades que compõem a camada. As classes `btFactory`, `btClient` e `btServer` implementam respectivamente as interfaces `iNetworkFactory`, `iNetworkClient` e `iNetworkServer`. Clientes de rede localizam dispositivos ao redor e utilizam a classe `device` para armazenar informações a respeito dos mesmos, que serão detalhadas na Seção 4.3). Tanto os objetos das classes servidor quanto cliente de rede utilizam os serviços da classe XXTEA quando há criptografia em suas comunicações (ativada durante a inicialização do *framework*).

A Figura 4.3 exemplifica uma chamada de rede entre os objetos servidor e cliente que executam em dispositivos diferentes. Quando um serviço no dispositivo A necessita enviar um comando de rede para o dispositivo B, ele executa `startClient`, responsável por enviar o comando de alto nível, passando como parâmetros o objeto serviço requisitante, o comando de alto nível a ser enviado ao dispositivo B, seus parâmetros (caso existam) e as informações de conexão do dispositivo. Após enviar o comando para

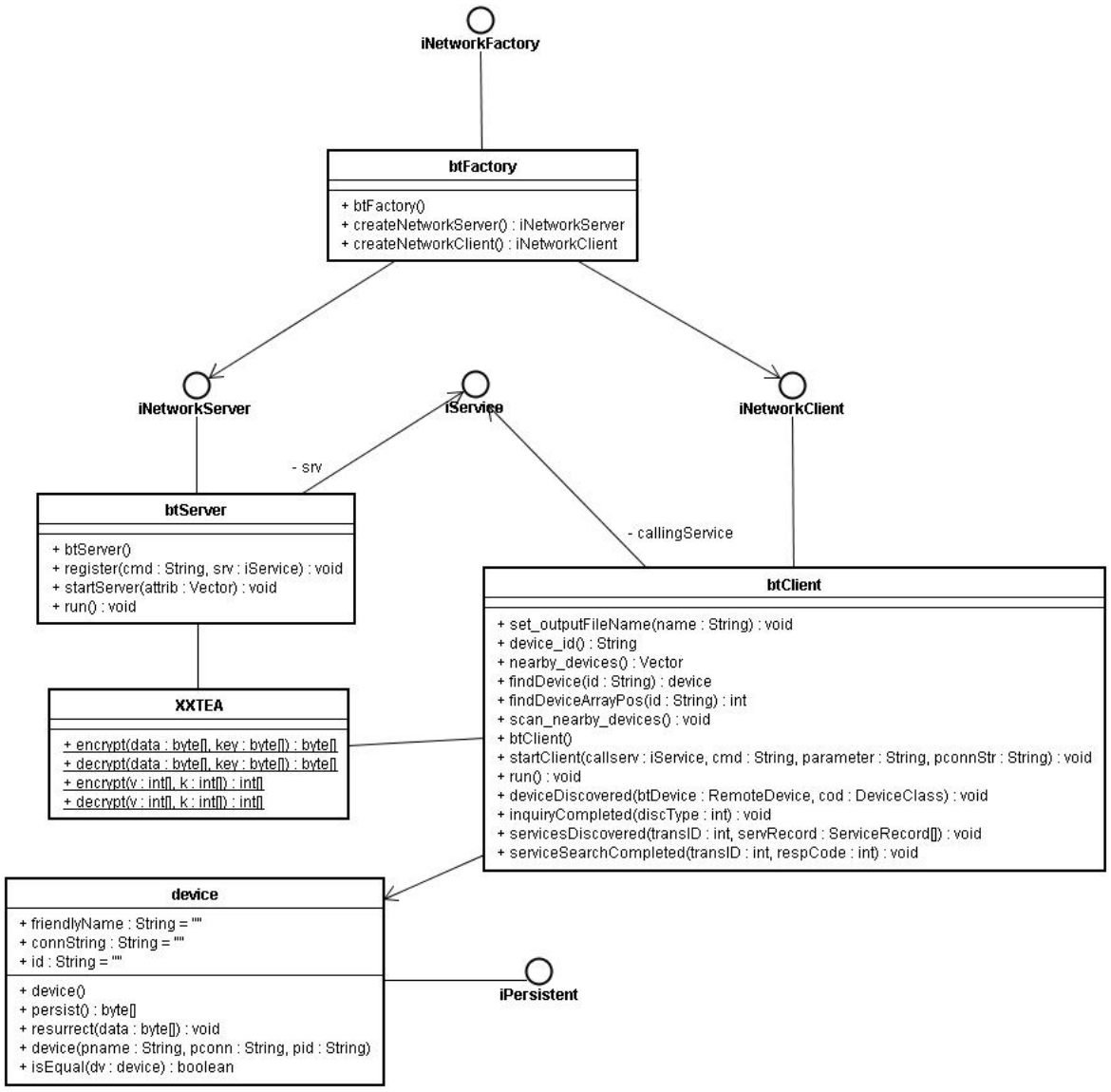


Figura 4.2: Diagrama dos componentes da Camada de Rede

o dispositivo B, o objeto de rede do dispositivo A envia uma mensagem para o serviço requisitante indicando a situação de envio do comando (sucesso ou falho no envio) e finaliza sua execução. Como a resposta é assíncrona e pode ocorrer a qualquer instante, o framework finaliza o processamento do objeto cliente de rede no dispositivo A e, através do seu objeto servidor de rede, aguarda pela resposta que será recebida e processada pelo método `netAnswer` do serviço em questão. No dispositivo B, o servidor de rede recebe a mensagem com o comando, localiza e repassa a mensagem para o serviço do *framework* responsável por processá-lo e envia sua resposta ao dispositivo A assim que a mesma estiver disponível.

Como forma de sempre disponibilizar uma lista atualizada de dispositivos ao

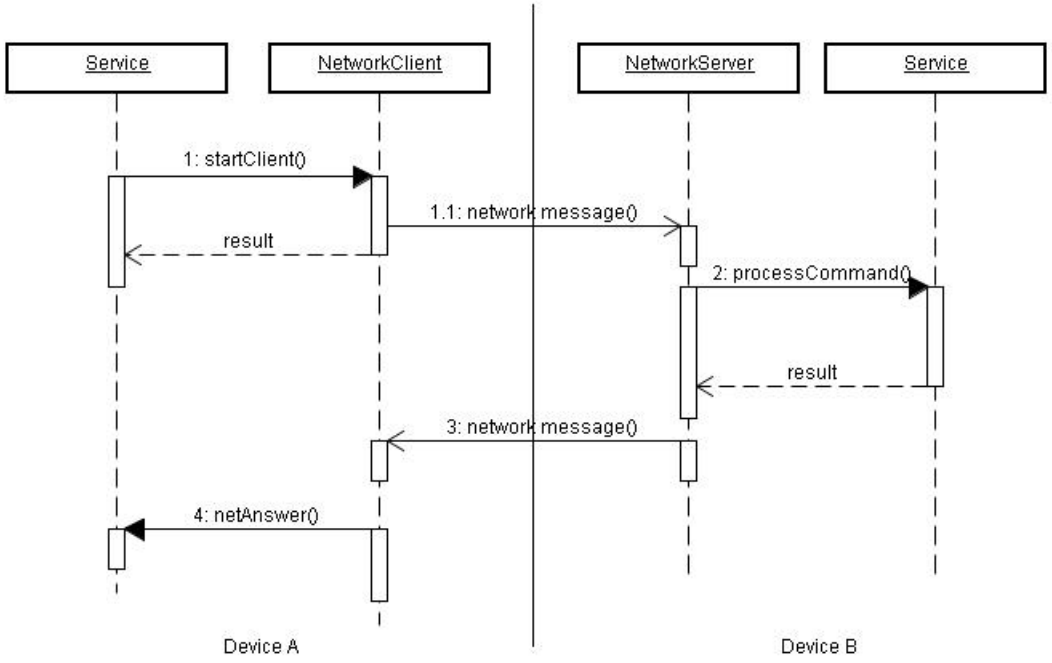


Figura 4.3: Exemplo de uma chamada de rede

redor, o *framework* possui uma *thread* com um cliente de rede agendada para ser executada em intervalos de tempo regulares a fim de iniciar um procedimento de varredura da vizinhança (`scan_nearby_devices()`) de forma transparente. Ao fim do procedimento de varredura o vetor de dispositivos ao redor é atualizado e a *thread* finaliza suas operações de forma silenciosa e sem intervenção do usuário. O intervalo de tempo entre varreduras é configurado durante a inicialização do *framework* e é aconselhável não utilizar tempos pequenos para evitar excessivas varreduras, o que acarretaria em um consumo excessivo de energia. Os custos envolvidos neste processo foram avaliados e serão apresentados na Seção 5.

4.3 A Camada de Serviços

A Camada de Serviços contém todos os serviços disponíveis no *framework*. Como mostra a Figura 4.1, esta é a única camada visível para a aplicação. Existem serviços que lidam com os contatos (com seus dispositivos) e grupos, compartilhamento de dados, contexto da aplicação e mensagens de texto. Como mostrado na Seção 4.2.1, todos os serviços implementam a interface `iService` e jamais acessam diretamente o subsistema de rede em uso, sempre fazendo uso das interfaces providas pela Camada de

Rede.

4.3.1 O Serviço de Contatos

O Serviço de Contatos implementa a noção de contatos e grupos do SNU. A filosofia de grupo deste serviço é reunir contatos em grupos que irão possuir as mesmas premissas de compartilhamento. Assim, é possível indicar que um determinado recurso está compartilhado com uma lista de usuários individualmente ou com um grupo.

Um contato possui um conjunto (possivelmente unitário) de dispositivos. Dispositivos móveis tendem a possuir um alto grau de envolvimento com seus usuários que costumam carregá-los consigo grande parte do tempo e raramente os compartilham com amigos [Bea05, ROPT05]. Desta forma, o SNU reconhece um dispositivo como um contato, porém entende que os contatos podem ter mais de um dispositivo. Sendo assim, cada contato possui uma lista de dispositivos vinculados, tornando possível o reconhecimento de um mesmo contato que utilize vários dispositivos.

Cada usuário é responsável por criar e gerenciar seus grupos localmente. Não há um servidor central ou qualquer outra solução distribuída de armazenamento e sincronização de grupos. Soluções centralizadas necessitam que pelo menos um nó funcione como servidor e soluções distribuídas necessitam que exista uma estreita relação de confiança entre os membros da rede, o que não pode ser garantido em redes *ad-hoc* espontâneas. Contudo, usuários poderem criar grupos e compartilhá-los publicamente, tornando possível sua transferência para outros dispositivos.

Todos os objetos do SNU (como grupos, itens de dados ou contextos) possuem duas listas de controle de acesso próprias, responsáveis por armazenar os grupos e os contatos que possuem permissão de leitura sobre eles. Apenas o proprietário do objeto pode alterá-lo. Sendo assim, não há comandos de rede para criação ou alteração de objetos remotos, apenas para leitura dos mesmos. Por exemplo, para compartilhar um objeto com João, este objeto deverá possuir o contato João em sua lista (de controle de acesso) de contatos ou possuir um grupo que contenha o contato João em sua lista (de controle de acesso) de grupos. Objetos podem ainda ser compartilhados de forma pública, ou seja, qualquer contato poderá acessá-los.

Este conceito de visibilidade através das listas de acesso facilita o compar-

tilhamento dos dados pelo usuário e está presente em todo o arcabouço. Usuários que solicitem informações compartilhadas só enxergam aquelas que passaram pela verificação de visibidade implementada pelas listas de acesso. Desta forma, usuários que não possuem seus contatos nas lista de acesso não conseguem se quer vislumbrar a existência de uma informação, garantindo, assim, a privacidade das mesmas.

A interface do Serviço de Contatos, `iContactService` é exibida na Listagem 4.5. O método `setContactServiceForm()` indica qual o formulário vinculado ao serviço que deverá receber as chamadas de retorno, uma vez que estas podem ocorrer de forma assíncrona. Como foi visto na Seção 4.2.2, para evitar longas esperas ou possíveis travamentos da aplicação, todos os comandos de alto nível transferidos pela rede são assíncronos. Desta forma, é necessário que cada serviço do *framework* indique qual é o formulário da aplicação do usuário que está vinculado ao mesmo, e onde serão exibidas informações relativas às respostas encaminhadas pela camada de rede.

Existem duas listas para armazenar os contatos e grupos gerenciados, recuperáveis pelos métodos `contacts()` e `groups()` respectivamente. Para a administração dos contatos, existem métodos para criar (`createContact()`), adicionar (`addContact()`), remover (`delContact()`), renomear (`renameContact()`) e atualizar (`updateContact()`) contatos. É bom observar que um contato deve ser criado a partir de um dispositivo localizado nas proximidades, ou seja, os contatos devem sempre possuir pelo menos um dispositivo vinculado. O *framework* identifica os contatos através dos seus dispositivos desta forma, contatos sem dispositivos não podem ser identificados.

Ainda no tocante aos contatos, existem métodos para localizá-los pelo seu identificador (`find_contact_by_id()`) ou pelos identificadores de dispositivos associados a eles (`find_contact_by_device_id()`). Estes identificadores de dispositivos são representados na implementação atual pelo endereço local *Bluetooth* (vide Seção 4.2.2).

De forma análoga, existem métodos para adicionar, remover, renomear, atualizar e localizar grupos (`addGroup()`, `delGroup()`, `renameGroup()`, `updateGroup()` e `findGroup()`, respectivamente). Como grupos são conjuntos de contatos, existem métodos para adicionar um contato a um grupo (`addContactToGroup()`) e remover um contato de um grupo (`delContactFromGroup()`).

```
1 package snu;
2
3 import java.util.Vector;
4
```

```
5  public interface iContactService {
6
7          void setContactServiceForm(iContactServiceForm param);
8
9          Vector groups();
10
11         group findGroup(String gname);
12
13         boolean addGroup(String gname);
14
15         boolean importGroup(int pos);
16
17         boolean delGroup(String gname);
18
19         int addContactToGroup(String gname, contact ct);
20
21         int delContactFromGroup(String gname, contact ct);
22
23         void getPublicGroupsFromDevice(int contactVectorIndex);
24
25         Vector imported_public_groups();
26
27         boolean addContact(contact ct, int pos);
28
29         boolean delContact(int pos);
30
31         boolean renameContact(String new_name, int pos);
32
33         Vector contacts();
34
35         contact find_contact_by_id(String id);
36
37         contact find_contact_by_device_id(String device_id);
38
39         void showScanResult();
40
41         contact createContact(device dv);
42
43         Vector ContactGroups(String ContactId);
44
45         boolean renameGroup(String actual_name, String new_name);
46
47         boolean updateContact(contact new_contact);
48
49         boolean updateGroup(group new_group);
50
51 }
```

Listagem 4.5: Interface iContactService

Também existem métodos para recuperar os grupos compartilhados de um determinado dispositivo (`getPublicGroupsFromDevice()`) e para acessar tais grupos (`imported_public_groups()`) após sua transferência. O *framework* não importa automaticamente os grupos compartilhados, apenas os transfere e disponibiliza à aplicação. Entende-se por importar, a operação de adição dos grupos recém transferidos a lista de grupos locais. Desta forma, o usuário pode decidir quais grupos recém transferidos deseja de fato importar, indicando-os individualmente através do método `importGroup()`.

Como explicado na Seção 4.2.1, toda classe que implementa um serviço do *framework* deve implementar a interface `iService`. Desta forma, a classe que implementa o serviço de contatos possui os métodos `processCommand()` e `netAnswer()` da

interface `iService`. `processCommand()` é utilizado para processar um comando recebido pela camada de rede enquanto que `netAnswer()` é utilizado para processar uma resposta a um comando previamente enviado. Como todo serviço implementa `iService`, a camada de rede localiza o objeto responsável pelo processamento dos comandos recebidos através da simples varedura da lista de comandos registrados pelo método `register()` e chama sempre os mesmos métodos (de `iService`) dos objetos vinculados aos comandos.

```
1  package snu;
2
3  public interface iContactServiceForm {
4
5    void showScanResult();
6
7    void showAvailablePublicGroups();
8
9  }
```

Listagem 4.6: Interface iContactServiceForm

A Listagem 4.6 exibe a interface `iContactServiceForm`, que deve ser implementada pela classe formulário vinculada ao serviço de contatos. Os formulários vinculados a cada serviço do *framework* devem implementar as interfaces apropriadas para receberem as chamadas de retorno dos comandos de rede enviados (vide Seção 4.2.2). Desta forma, `showScanResult()` é chamado assim que o *framework* recebe a resposta a um comando de rede para localizar explicitamente[1] todos os dispositivos ao redor, enquanto que `showAvailablePublicGroups()` é utilizado para indicar o recebimento da mensagem contendo todos os grupos compartilhados por um dado dispositivo.

Implementação

No relacionamento entre as classes do Serviço de Contatos mostrado na Figura 4.4 é possível observar que o serviço de contatos é composto por objetos contatos, que por sua vez são obrigatoriamente compostos por objetos dispositivos, uma vez que cada contato tem que possuir pelo menos um dispositivo vinculado. Os grupos são formas de reunir contatos e, portanto podem existir ou não.

A classe grupos possui uma lista de contatos e métodos para adicionar, remover e verificar a existência de contatos (`addContact()`, `delContact()` e `hasContactByID()`,

[1] A localização explícita é aquela iniciada pelo usuário, enquanto que a implícita é executada transparentemente pelo *framework*, acionada em uma freqüência pré-especificada durante sua inicialização (vide Seção 4.2.2).

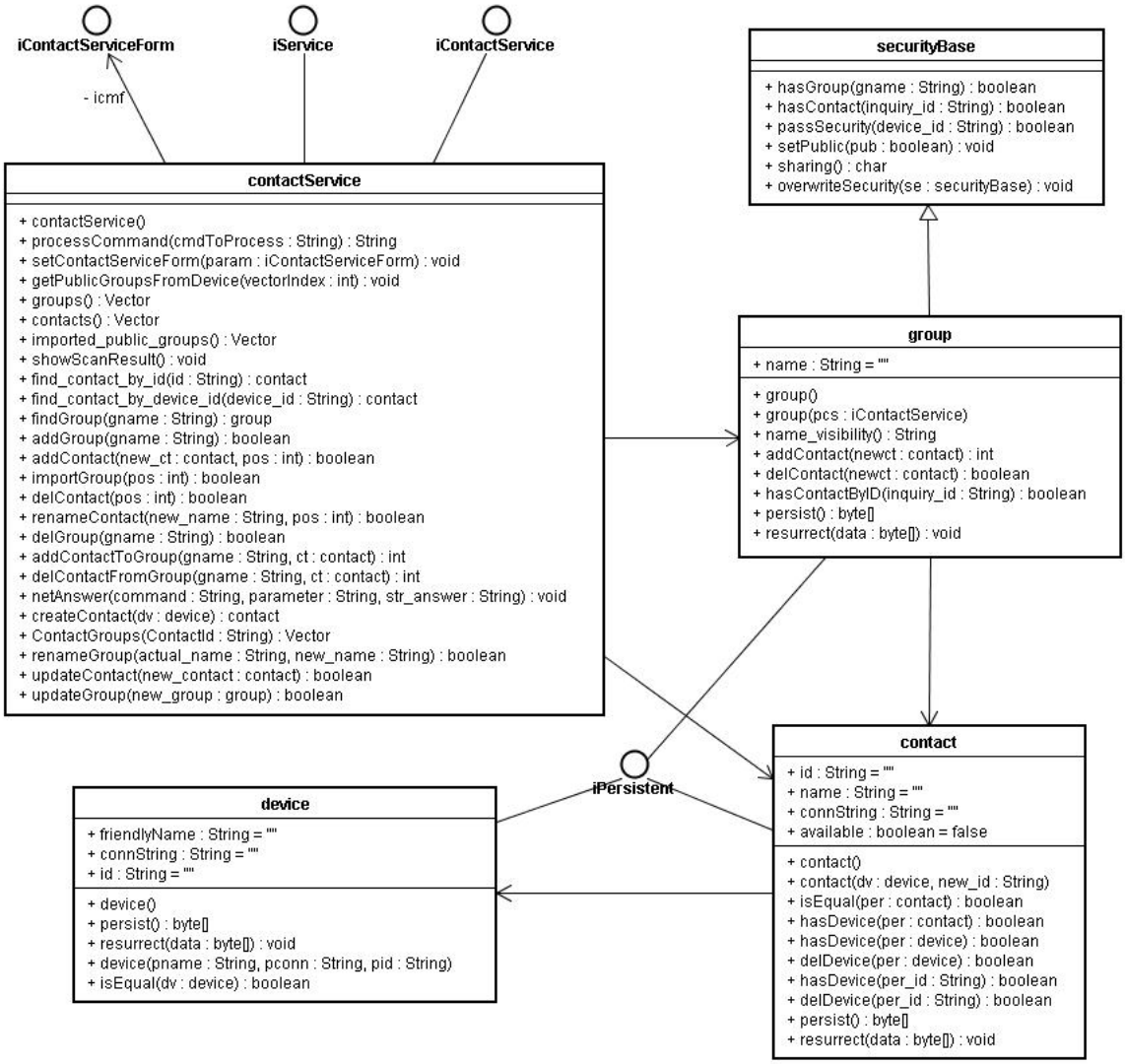


Figura 4.4: Componentes do Serviço de Contatos

respectivamente). Por sua vez, a classe contatos possui uma lista de dispositivos e métodos para testar a existência de um dispositivo associado ao contato (`hasDevice()`) e para removê-lo (`delDevice()`). Como cada contato possui uma lista de dispositivos vinculados, o método `hasDevice()` pode ser utilizado para testar se um contato já possui um dado dispositivo vinculado.

Ao adionar um contato, não há a necessidade de um método `addDevice()`, uma vez que durante a criação de um contato deve-se obrigatoriamente passar um dispositivo como parâmetro. Para adicionar um segundo dispositivo ao um contato existente, basta criar um novo contato a partir do dispositivo a ser adicionado e chamar o método `addContact()` (serviço de contatos) passando o novo contato e a posição do contato existente (aquele que receberá outro dispositivo) no respectivo vetor do serviço de contatos. A classe dispositivos possui apenas informações relevantes ao seu acesso: seu nome amigável

(`friendlyName`), um identificador de rede (`id`) e dados de conexão (`connString`).

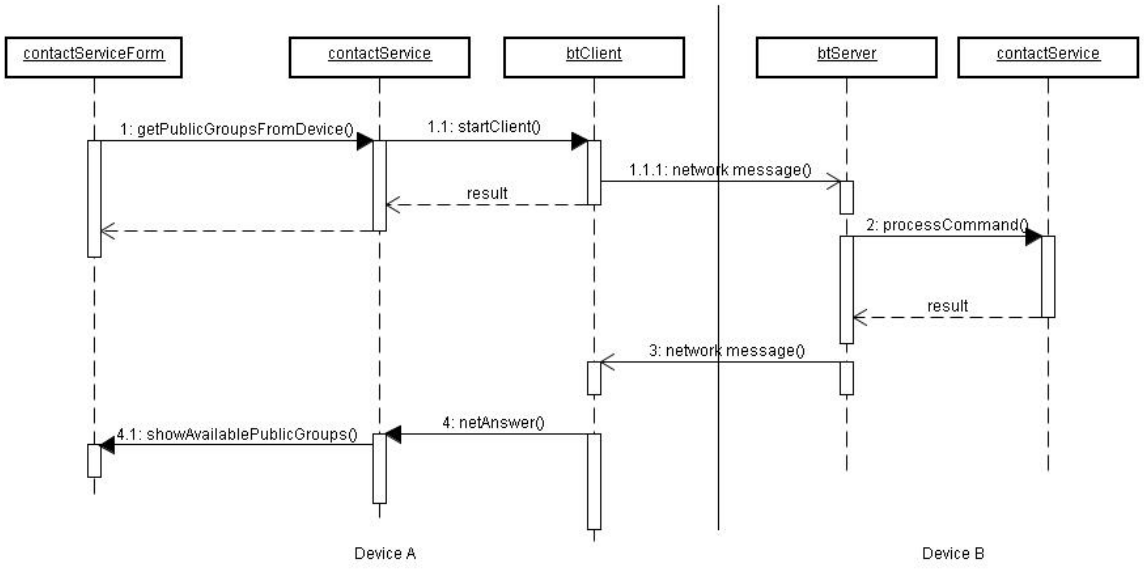


Figura 4.5: Exemplo de uma chamada *callback*

Como forma de exemplificar a interação entre o Serviço de Contatos e Camada de Rede, a Figura 4.5 exibe o diagrama de seqüência de uma chamada de rede para recuperar os grupos públicos compartilhados, na figura, pelo usuário no dispositivo B. No dispositivo A, o formulário da aplicação do usuário `contactServiceForm`, que implementa a interface `iContactServiceForm`, chama `getPublicGroupsFromDevice()` do Serviço de Contatos (`contactService`), descrito em `iContactService`. O Serviço de Contatos, utilizando o cliente de rede implementado por `btClient`, envia o comando "`PublicGroups`"(registrado durante a inicialização do serviço) para o servidor de rede implementado por `btServer` no dispositivo B. O servidor de rede no dispositivo B recebe o comando, verifica qual objeto pertencente à camada de serviços do *framework* deve processá-lo e faz uma chamada ao seu método `processCommand()`, definido em `iService`. Após seu processamento, o serviço responsável envia a resposta de volta ao servidor de rede que a remete em forma de uma seqüência de *bytes* de volta ao dispositivo A. Este, for sua vez, recebe a resposta, a interpreta e executa o método `netAnswer()` de `contactService`) que atualiza seus vetores e executa `showAvailablePublicGroups()` disponível em `contactServiceForm` para exibir, no dispositivo A, os resultados ao usuário.

Implementação da Persistência

Todas as classes que necessitam armazenar informações persistentes possuem

os métodos de serialização (`persist()`) e reconstrução (`resurrect()`) definidos na interface `iPersistent`, Listagem 4.7. O método `persist()` deve retratar o estado do objeto retornando uma seqüência de *bytes* com os valores das propriedades do mesmo, de tal forma que possa ser reconstituído posteriormente. `resurrect()` recebe uma seqüência de *bytes* gerada por `persist()` e recarrega as propriedades do objeto em questão, restaurando-o ao estado retratado.

Uma vez serializadas, as informações são armazenadas com a utilização do serviço J2ME RMS (*Record Management Service*), disponível em todas as versões do J2ME. RMS é um serviço seguro de armazenamento de registro de qualquer tamanho, porém sem a definição de campos. O *framework* SNU utiliza um registro RMS para representar cada objeto serializado e utiliza um repositório RMS para cada classe que necessite de persistência. Assim, existem repositórios para contatos (com seus dispositivos), grupos, itens de dados e contextos. O resultado de `persist()` é armazenado sem alterações via RMS.

```
1  package snu;
2
3  public interface iPersistent {
4
5      byte[] persist() throws IOException;
6
7      void resurrect( byte[] data ) throws IOException;
8
9  }
```

Listagem 4.7: Interface iPersistent

Implementação da Segurança

As listas de controle de acesso citadas na Seção 4.3.1, assim como seus métodos de gerenciamento, foram implementados na classe `securityBase`, Listagem 4.8, estendida por todos os objetos de SNU que necessitam deste controle. A classe `securityBase` define os vetores públicos `groupNames` e `contactsIDs` que armazenam, respectivamente, o nome dos grupos e os identificadores locais dos contatos. Não é permitido existir dois ou mais grupos com o mesmo nome. Portanto, o nome dos grupos deve ser único em cada dispositivo. Como é bem possível que haja mais de um contato com o mesmo nome, `securityBase` armazena um identificador local para cada contato no vetor público ao invés de seu nome.

```
1  package snu;
2
```

```
3  import java.util.Vector;
4
5  public class securityBase{
6
7          public Vector groupNames = new Vector();
8
9          public Vector contactsIDs = new Vector();
10
11         public boolean hasGroup(String gname);
12
13         public boolean hasContact(String inquiry_id);
14
15         public boolean passSecurity(String device_id);
16
17         public void setPublic(boolean pub);
18
19         public char sharing();
20
21     public void overwriteSecurity(securityBase se);
22
23 }
```

Listagem 4.8: Assinatura da Classe securityBase

Também são disponibilizados métodos para verificar se um grupo ou contato está presente nas listas de acesso (`hasGroup()` e `hasContact()`, respectivamente). Desta forma, é possível testar se um dado grupo ou contato já faz parte das listas de controle de acesso. Para verificar se um dispositivo tem permissão de acesso utiliza-se o método `passSecurity()`, passando o identificador único do dispositivo como parâmetro. Este método verifica se o identificador de dispositivo repassado está vinculado a algum contato pertencente a lista de contatos ou a algum contato pertencente a um dos grupos listados na lista de grupos.

Para limpar as listas de controle de acesso, tornando o objeto totalmente privado (ou público) utiliza-se o método `setPublic()` com um parâmetro boleano (verdadeiro para público e falso para privado). O método `sharing()` retorna a situação do objeto com relação ao seu compartilhamento. Os possíveis valores de retorno deste método são: '-', caso o objeto seja privado, 'P', caso seja público, 'C', caso seja compartilhado com algum contato, 'G' com algum grupo ou 'S' com ambos. Este método é de grande valia quando se está desenvolvendo a interface com o usuário, para, por exemplo, utilizá-lo para indicar graficamente o tipo de compartilhamento presente.

Por fim, temos o método `overwriteSecurity()`, que sobreescreve as listas de acesso utilizando outro objeto como base, limpando as lista de acesso atuais e copiando as lista de acesso disponíveis no objeto base.

4.3.2 O Serviço de Dados

O Serviço de Dados do SNU disponibiliza uma forma de compartilhar dados sem a necessidade de um controle centralizado. São utilizados os conceitos de itens de dados e coleções. Um item de dado é um par de *strings* representando o nome e o valor do item, porém sem formatação. Caso o item de dado seja um arquivo qualquer, seu valor será sempre o caminho local do arquivo representado. Uma coleção é um conjunto de itens de dados. Uma coleção pode ser composta por coleções, de forma semelhante a um sistema de arquivos onde diretórios podem conter além de arquivos, outros diretórios. Sempre existe pelo menos uma coleção inicial (raiz) de onde nascem as demais e onde pode-se armazenar qualquer quantidade de itens de dados. Desta forma, as coleções possuem métodos direcionados ao gerenciamento de seus itens de dados.

Itens de dados podem ser utilizados para armazenar informações pessoais, como nome, endereço ou telefone, ou ainda informações profissionais, como o local de trabalho, a função ou profissão. Também é possível armazenar informações relativas a serviços ou produtos como um cardápio ou os custos de uma corrida de táxi. Não há limite para o número de itens de dados criados ou para o tamanho dos arquivos representados por eles[2]. Desta forma, é possível compartilhar fotos, músicas, vídeos ou arquivos texto, como um *curriculum vitae*. Todos estes itens de dados podem ser organizados em coleções de qualquer tamanho ou profundidade (coleções dentro de coleções). Usuários podem criar um tipo de portfólio profissional, armazenando fotos e diagramas de seus trabalhos, carregando-os consigo e compartilhando-os a todo instante.

Como itens de dados e coleções são objetos do SNU, eles seguem os critérios de segurança dos grupos, vistos na Seção 4.3.1. Ou seja, quando um usuário solicita a lista de itens de dados ou coleções compartilhadas, receberá apenas aquelas que foram aprovadas pela verificação de segurança (definidas em `securityBase`). Entretanto, quando um item de dado representa um arquivo, este não será transferido de imediato, uma vez que esta operação pode facilmente tomar mais tempo e espaço que o disponível pelo usuário. Nesta situação, o *framework* tranfere os itens de dados indicando que se tratam de um arquivo e a aplicação deverá solicitar a transferência individual dos arquivos indicados, desta forma, poderá controlar as esperas e complicações derivadas de um processo de importação de arquivos (arquivos com mesmo nome, grandes demais, incapazes de serem visualizados,

[2]Limitado apenas pela memória de massa do dispositivo.

etc...).

Como coleções podem ser compostas por outras coleções, e não há limite para o número de níveis ou de itens de dados em cada nível, existe a necessidade de se evitar as longas transferências que fatalmente ocorreriam caso grandes coleções fossem compartilhadas. Longas transferências podem gerar esperas prolongadas ou indefinidas, assim como podem consumir grandes quantidades de energia pela utilização continua do enlace de rádio. Como forma de evitar tais problemas, todas as transferências de itens de dados são feitas até uma dada profundidade *N*. Ou seja, caso haja uma coleção composta de 5 coleções em 3 níveis diferentes, apenas as N primeiras coleções e seus itens de dados serão transferidos. Cabe à aplicação do usuário solicitar a importação das demais coleções em profundidade, sempre em blocos de profundidade *N*. O valor padrão para N é 2. Contudo, este valor pode ser livremente alterado durante a inicialização do *framework*. Para determinar individualmente qual coleção deseja transferir, o SNU utiliza uma nomenclatura totalmente qualificada, como em *collection1.collection2.collection3*.

Caso um usuário faça uma cópia local de qualquer item de dado compartilhado, este se tornará responsável por futuras atualizações do mesmo, uma vez que o SNU não provê métodos de atualização automática ou controle de versões. Porém, para facilitar esta tarefa, são providos métodos que calculam os valores da função *hash* MD5 (*Message-Digest 5* [Riv92]) de qualquer item de dado, seja um simples *string* ou um arquivo de qualquer tamanho.

```
1  package snu;
2
3  import java.util.Enumeration;
4
5  public interface iDataService {
6
7          void setDataServiceForm(iDataServiceForm fparam);
8
9          void netGetPublicDataFromContact(int device_pos);
10
11         void netGetFileDataFromContact(
12                 int device_pos, String diName, String fileName);
13
14         void netGetFileMD5(int device_pos, String diName);
15
16         void netGetCollectionFromContact(int device_pos, String cname);
17
18         Enumeration get_LocalFiles(String path);
19
20         collection getRoot();
21
22         void saveCollections();
23
24         collection foreignSharedCollections();
25
26 }
```

Listagem 4.9: Interface iDataService

A Listagem 4.9 apresenta a interface do Serviço de Dados. Seguindo a mesma lógica do Serviço de Contatos, há um método para indicar qual formulário do usuário deverá receber as chamadas de retorno da camada de rede (`setDataServiceForm()`).

Em princípio existe pelo menos a coleção raiz (como em um sistema de arquivos convencional), de onde nascerão as demais coleções. Não há limites para a quantidade de coleções ou itens de dados por coleção. O método `getRoot()` retorna a coleção raiz e todos os itens de dados nela presentes.

Quando um usuário A deseja recuperar os itens de dados e coleções compartilhados por um contato B, ele primeiramente utiliza `netGetPublicDataFromContact()` para recuperar os dados públicos compartilhados pelo mesmo. Neste momento, o serviço de dados do contato B verifica quais dados são acessíveis pelo dispositivo que solicitou a transferência dos dados públicos (dispositivo do usuário A). Em seguida, o serviço de dados responde com uma coleção base composta pelos itens de dados e coleções acessíveis pelo contato A. Portanto, cada vez que um contato solicita a transferência dos itens de dados públicos de outro, ele obterá uma coleção que representa a visão a ele permitida.

Os dados compartilhados recém transferidos são acessados através do método `foreignSharedCollections()`. Como as coleções transferidas podem ter sido truncadas numa determinada profundidade, o método `netGetCollectionFromContact()` recupera individualmente uma coleção qualquer através de seu nome qualificado (col1.col2.col3), indicando a hierarquia em questão. Os itens de dados e coleções recebidos não são imediatamente importados para a estrutura local do usuário. Cabe a aplicação do usuário escolher e adicionar as informações de interesse.

O método `netGetFileDataFromContact()` é utilizado para solicitar a transferência de um arquivo compartilhado também utilizando seu nome qualificado. Para solicitar o cálculo da função *hash* MD5 de um dado arquivo compartilhado no dispositivo remoto (auxilia no controle de versão) tem-se o método `netGetFileMD5()`. É importante salientar que o *framework* é capaz de calcular o valor da função *hash* MD5 para arquivos de qualquer tamanho, independentemente da memória principal do dispositivo, uma vez que utiliza *buffers* no seu processamento.

Como forma de auxiliar no desenvolvimento de aplicações que acessem o sistema de arquivos local (JSR 75) [II04], a interface provê o método `get_LocalFiles()`, responsável por retornar um objeto `Enumeration` contendo todas as entradas (arquivos

ou outros diretórios) em um dado diretório. Este método não é utilizado pelos serviços do `famework`, porém facilita o desenvolvimento de aplicações que compartilhem itens de dados que representem arquivos locais, uma vez que já implementa o acesso e navegação na estrutura do sistema de arquivos local.

```
1  package snu;
2
3  public interface iDataServiceForm {
4
5          void showAvailableForeignPublicData();
6
7          void showImportedFile(String fileName);
8
9          void valueOfMD5(String md5);
10
11         void showForeignCollection(collection col);
12 }
```

Listagem 4.10: Interface iDataServiceForm

A Listagem 4.10 apresenta a interface do formulário do usuário vinculado ao Serviço de Dados. Existem quatro métodos que correspondem à ação que deve ser tomada pela aplicação do usuário em resposta aos comandos de rede descritos anteriormente. `ShowAvailableForeignPublicData()` é chamado quando o *framework* recebe a coleção compartilhada de um dado dispositivo após solicitar sua transferência; `showImportedFile()` é chamado ao término da transferência de um arquivo; o método `valueOfMD5()` é chamado com o valor da função *hash* MD5 de um item de dado compartilhado remotamente; e `showForeignCollection()` é chamado logo após o término da transferência de uma coleção compartilhada.

Implementação

A Figura 4.6 exibe o relacionamento entre os objetos que compõem o Serviço de Dados. Como esperado, o Serviço de Dados deve implementar as interfaces `iDataService` e `iService` e, portanto, provê todos os métodos nelas declarados. Também existe uma referência à interface `iDataServiceForm` para o processamento das respostas dos comandos de rede. A interface do serviço não possui métodos de interação com os itens de dados, uma vez que estes são totalmente manipulados pelas coleções. Tanto coleções como itens de dados implementam a interface `iPersistent`, pois são objetos persistentes no *framework*. Assim como ocorre com os grupos compartilhados, as coleções e os itens de dados também estendem a classe `securityBase`, a fim de disponibilizar seu suporte ao compartilhamento.

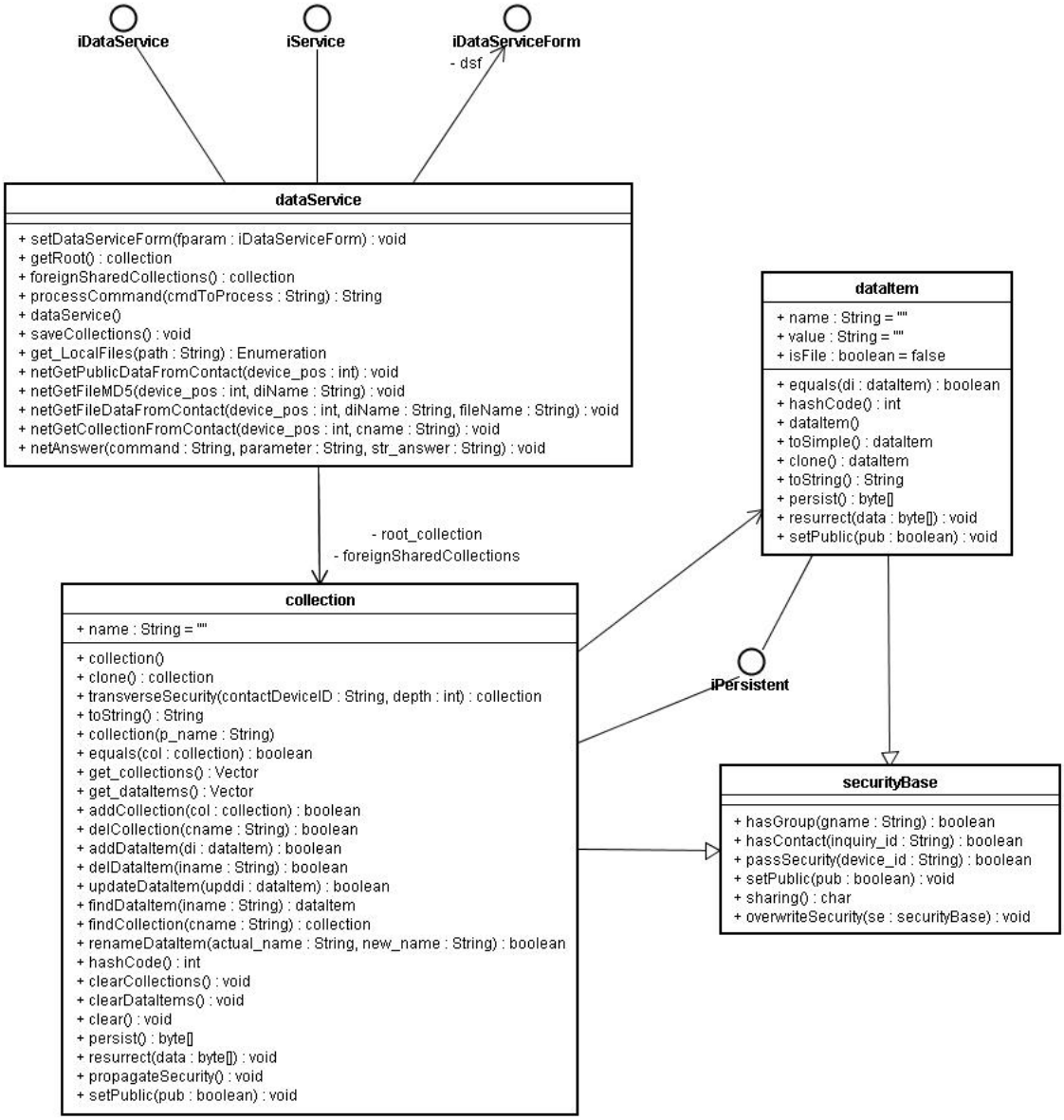


Figura 4.6: Componentes do Serviço de Dados

As coleções possuem métodos para adicionar uma coleção (`addCollection()`) ou removê-la (`delCollection()`), remover todas as coleções (`clearCollections()`), remover todos os itens de dados (`clearDataItems()`), remover todas as coleções e itens de dados (`clear()`), localizar uma coleção pelo seu nome (`findCollection()`) e calcular uma função *hash* de si própria (`hashCode()`). No tocante aos itens de dados, existem métodos para adicionar (`addDataItem()`), remover (`delDataItem()`), atualizar (`updateDataItem()`), renomear (`renameDataItem()`) e localizar (`findDataItem()`, pelo nome) um item de dado.

Também há métodos para gerar um clone da própria coleção (`clone()`), compará-la com outra coleção (`equals()`), assim como torná-la pública (`setPublic()` atualiza

a coleção e todos os objetos pertencentes a ela, tornando-a completamente pública). Para replicar as listas de acesso de uma coleção para todos os seus objetos utiliza-se o método `propagateSecurity()`. Para gerar uma coleção que possua apenas os objetos com permissão de leitura para um determinado contato (através do identificador de seu dispositivo) até uma determinada profundidade, utiliza-se o método `transverseSecurity()`. Este método é utilizado pelo *framework* quando o serviço de dados recebe o comando `foreignSharedCollections()` para retornar apenas as coleções compartilhadas com o dispositivo solicitante.

4.3.3 O Serviço de Contexto

O Serviço de Contexto foi projetado para compartilhar informações de contexto da aplicação, como o humor do usuário ou o volume do seu toque (o que pode indicar sua disponibilidade). Contudo, também pode ser utilizado para compartilhar informações de contexto do dispositivo [ROPT05], como memória de massa ou bateria disponível, perfil do usuário ou localização. Este serviço é especialmente útil para auxiliar o processo decisório da aplicação do usuário assim que uma dada condição for alcançada. Por exemplo, uma aplicação pode esperar até que o volume do toque de um determinado usuário não esteja mais mudo (quando o usuário está potencialmente disponível) para enviar-lhe uma mensagem de texto solicitando o início de uma sessão de bate-papo; ou pode reduzir a resolução de um *stream* de vídeo porque o processo de compressão está consumindo muita energia.

Algumas informações de contexto devem ser explicitamente fornecidas pelo usuário, como seu humor ou sua disponibilidade. Já outras podem ser lidas diretamente a partir do dispositivo, como sua memória disponível, condição de bateria ou volume do toque. Contudo, o *framework* disponibiliza apenas a infra-estrutura para criação, notificação e compartilhamento de contexto do usuário, não existindo funções que recuperem informações específicas do dispositivo, como as citadas.

O compartilhamento das informações de contexto é feito através das mesmas regras adotadas nos demais objetos do *framework*. Contextos são objetos do SNU e, portanto, estendem a classe `securityBase` possuindo as mesmas listas de acesso presente nos grupos de contatos e itens de dados. Para ser notificado da alteração de um determinado contexto, o contato deve primeiramente pertencer a umas dessas listas de acesso.

Quando um usuário deseja ser notificado sobre a alteração de uma determinada informação de contexto remota, ele deve enviar uma mensagem ao dispositivo em questão para se subscrever no contexto desejado. Assim que um dado contexto sofre atualização, o *framework* envia uma mensagem de atualização para todos os dispositivos subscritos. Caso um dispositivo subscrito não esteja mais disponível, ele permanecerá na lista até que solicite sua retirada explicitamente ou até que a aplicação seja finalizada. Como o *framework* não pode prever quando haverá falha de conexão[3] com os dispositivos ao redor, eles permanecem na lista para futuras tentativas.

Os cenários *ad-hoc* com que trabalhamos nos levam a crer que as informações de contexto (não o contexto em si, apenas seu estado, por exemplo o contexto humor pode ter os valores alegre ou triste) só são úteis em determinados intervalos de tempo. Veja o seguinte exemplo: quando adolescentes no intervalo entre aulas se reúnem no pátio da escola, podem utilizar uma aplicação de bate-papo para fazer novos amigos. Sendo assim, indicam que sua situação é "disponível" e compartilham essa condição com qualquer outro contato. Nesse momento é conveniente iniciar conversas com novas pessoas. Porém, ao termino do intervalo, quando usualmente se encerraria a aplicação de bate-papo, não será mais relevante armazenar a situação "disponível". Isto ocorre porque na próxima vez em que essa aplicação for iniciada, não há garantias de que essa situação estará correta. Portanto, consideramos que tais informações não necessitam de persistência.

```
1  package snu;
2
3  import java.util.Vector;
4
5  public interface iContextService {
6
7          void setContextServiceForm(iContextServiceForm fparam);
8
9          boolean addContext(context new_cnt);
10
11         boolean delContext(String cnt_name);
12
13         boolean updateContext(context new_cnt);
14
15         context findContext(String cnt_name);
16
17         Vector get_contexts();
18
19         Vector get_subscriptions();
20
21         void netSubscribeDevice(int dev_pos, String context);
22
23         void netUnsubscribeDevice(int dev_pos, String context);
24
25 }
```

Listagem 4.11: Interface iContextService

[3]Em ambientes com muitas pessoas ou mobílias, pequenas alterações na posição do dispositivo podem causar falhas de conexão em redes de rádio de curto alcance.

Na Listagem 4.11 encontramos a interface do Serviço de Contexto. Além do método que indica o formulário do usuário responsável pelo processamento das resposta aos comandos de rede (`setContextServiceForm()`), existem os métodos para adicionar (`addContext()`), remover (`delContext()`) e atualizar (`updateContext()`) contextos. O método `findContext()` localiza um contexto a partir do seu nome. Também há métodos para recuperar todos os contextos existentes (`get_contexts()`), assim como recuperar todas as subscrições existentes (`get_subscriptions()`). As subscrições indicam quais dispositivos desejam ser notificados quando há mudança de valor do contexto em questão.

No tocante aos métodos de subscrição, existe o método para solicitar uma subscrição em um determinado contexto (`netSubscribeDevice()`) e para solicitar o cancelamento da mesma (`netUnsubscribeDevice()`). Ambos os métodos recebem como parâmetros o dispositivo no qual deseja se subscrever (indicando sua posição no vetor de dispositivos próximos), assim como o contexto em questão (maiores detalhes na seção de implementação a seguir).

```
1  package snu;
2
3  public interface iContextServiceForm {
4
5          void subscribeDevice_answer(boolean ok);
6
7          void unsubscribeDevice_answer(boolean ok);
8
9          void updateContext_answer(boolean ok);
10
11         void update_notification(subscription sb);
12
13 }
```

Listagem 4.12: Interface iContextServiceForm

A Listagem 4.12 exibe a interface do formulário da aplicação do usuário que deverá processar os resultados dos comandos de rede. `subscribeDevice_answer()` e `unsubscribeDevice_answer()` são métodos que retornam simples respostas booleanas aos comandos de subscrição e cancelamento da mesma, respectivamente. Quando um contexto sofre atualização, o *framework* automaticamente percorre a lista de dispositivo subscritos e envia uma mensagem com a alteração do contexto para cada um. Ao receber essa mensagem, o dispositivo subscrito executa o método `update_notification()` repassando a alteração realizada e, em resposta, envia uma mensagem de volta, indicando sucesso ou falha (`updateContext_answer()`) durante o processo de atualização local.

Implementação

A Figura 4.7 exibe o relacionamento entre os objetos que compõem o Serviço de Contexto. Este serviço deve implementar a interface `iContextService` e, assim como no restante dos serviços (vide Seção 4.2.1), a interface `iService`. Como ocorre nos demais serviços, existe uma referência à interface `iContextServiceForm` para o processamento das respostas dos comandos de rede. O serviço possui listas dos contextos criados e dos dispositivos subscritos. Os contextos implementam a interface `iPersistent` pois são objetos persistentes no *framework*, assim como também estendem a classe `securityBase` a fim de prover o suporte ao compartilhamento.

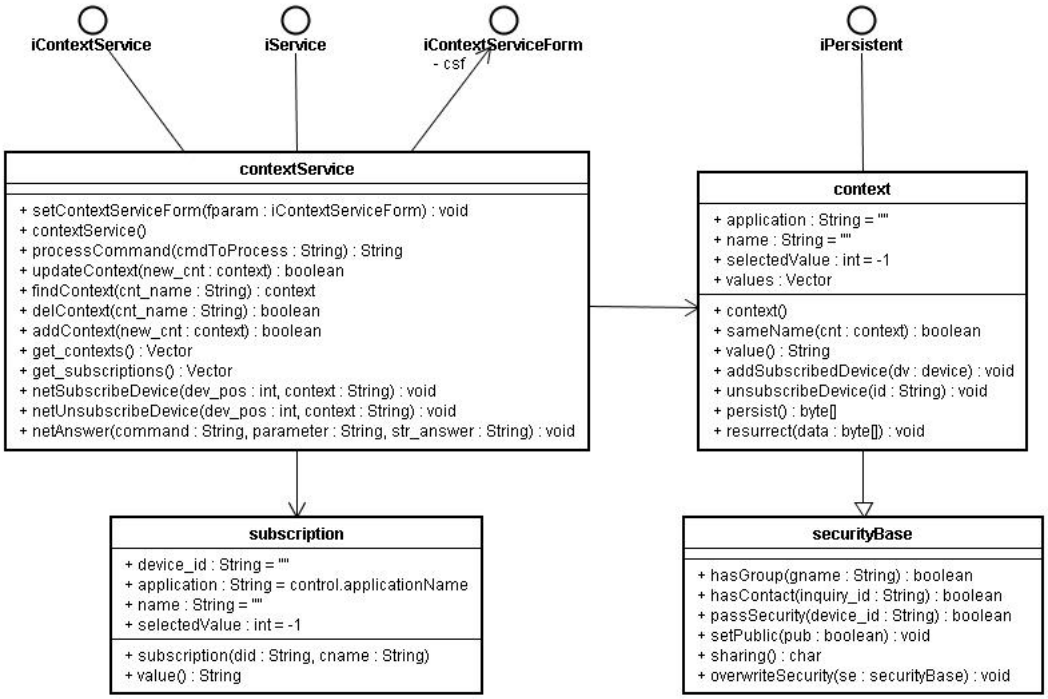


Figura 4.7: Componentes do Serviço de Contexto

Os objetos contexto possuem quatro propriedades: o nome da aplicação que criou o contexto, o nome do contexto, um vetor com seus possíveis valores e o índice (do vetor de valores) de seu valor atual. Existem métodos para comparar contextos (`sameName()`), verificando se os contextos possuem os mesmos nomes da aplicação e do contexto, retornar o valor atual do contexto (`value()`), assim como subscrever e cancelar a subscrição de um dado dispositivo (`addSubscribedDevice()` e `unsubscribeDevice()`, respectivamente).

Os objetos de subscrição armazenam as informações dos dispositivos e contex-

tos subscritos e possuem quatro propriedades públicas: o dispositivo detentor do contexto, o nome da aplicação que criou o contexto, o nome do contexto, e o índice (da lista de valores) de seu valor atual. Como os contextos devem ser obrigatoriamente conhecidos por ambos os dispositivos envolvidos no processo de subscrição, pois pertencem à mesma aplicação, não há necessidade de armazenar a lista dos valores do contexto no objeto subscrição. Esta lista já está disponível na própria definição do contexto. O método `value()` do objeto de subscrição recupera o valor do contexto subscrito a partir da lista de valores do contexto e da propriedade `selectedValue` do objeto.

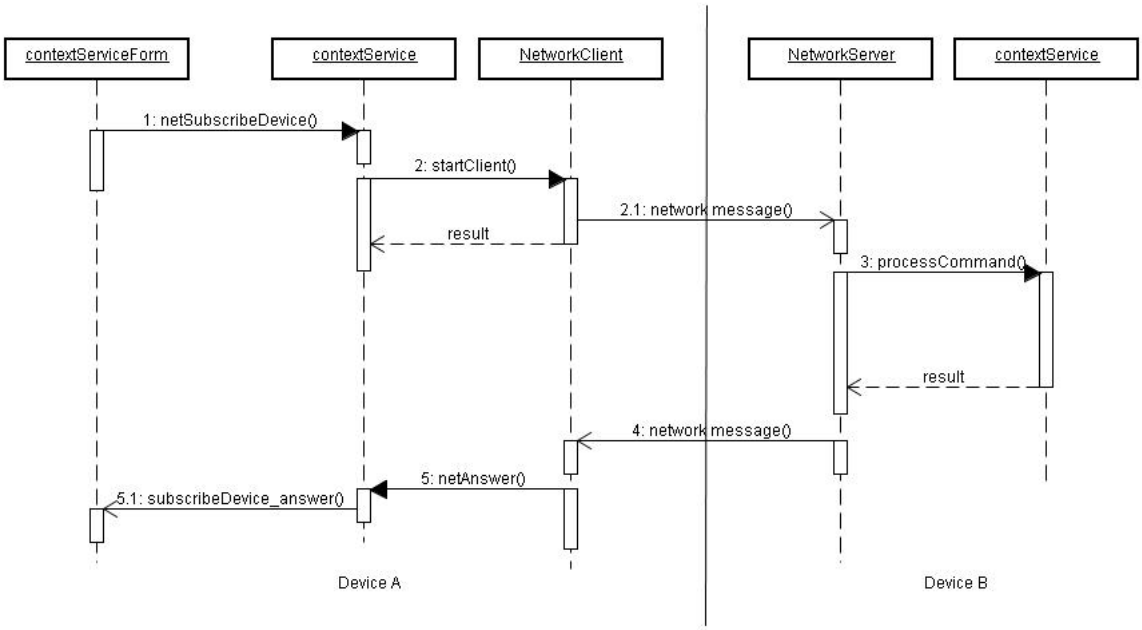


Figura 4.8: Diagrama de seqüência de um pedido de subscrição de contexto

A Figura 4.8 mostra o diagrama de seqüência de um pedido de subscrição de contexto feito pelo dispositivo A ao dispositivo B. Supomos que o usuário portador do dispositivo A esteja em um bar e deseje fazer novos amigos. Ele ativa sua aplicação de bate-papo e indica que sua situação é disponível. Outro usuário (com o dispositivo B) está no mesmo bar e utiliza a mesma aplicação de bate-papo, mas indicou que sua condição atual é ocupado, pois acaba de encontrar alguns conhecidos que a muito não via. Ao localizar o usuário B, o usuário A solicita subscrição no seu contexto "situação" (passo 1 da Figura 4.8). A mensagem é então transmitida pela camada de rede ao dispositivo do usuário B (passo 2). No dispositivo B a mensagem é repassada para o Serviço de Contexto (passo 3) que atualiza a lista de subscrições do contexto em questão e responde a solicitação (passo 4). A resposta é enviada de volta ao Serviço de Contexto do dispositivo A (passo 5) que faz uma chamada ao método `subscribeDevice_answer()` repassando o

resultado da solicitação. Desta forma, assim que o contexto em B for alterado, o usuário A será notificado.

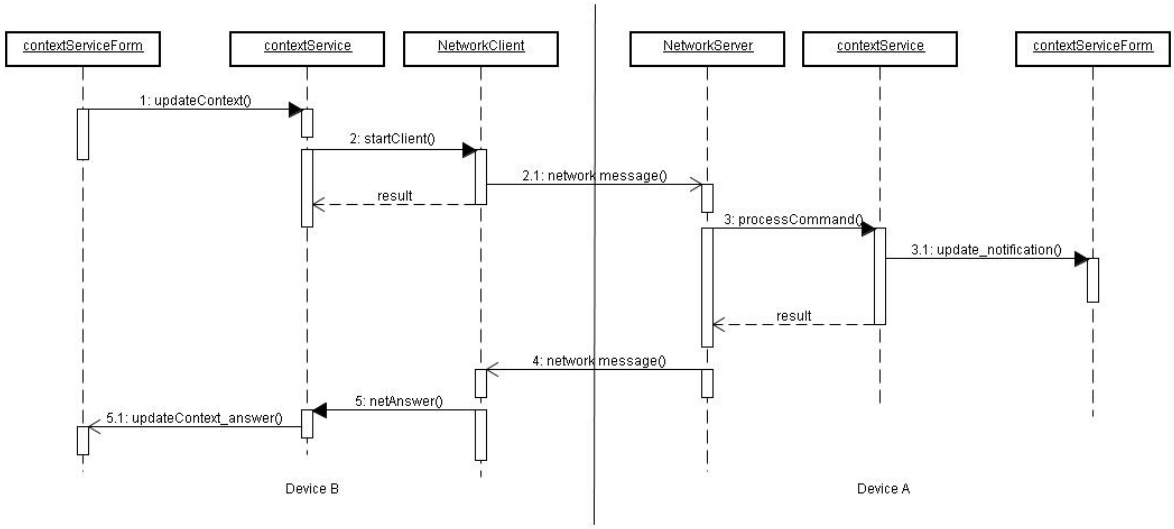


Figura 4.9: Diagrama de seqüência de uma atualização de contexto

Seguindo o exemplo anterior, a Figura 4.9 mostra o diagrama de seqüência de uma atualização de um contexto pertencente ao dispositivo B (usuário que estava conversando com amigos que a muito não via) e subscrito pelo dispositivo A (usuário disponível que deseja ser notificado quando B estiver livre). Ao atualizar um contexto, o Serviço de Contatos no dispositivo B envia uma mensagem de rede para o dispositivo A subscrito no contexto em questão que, ao receber a informação, atualiza o objeto `subscription` correspondente e faz uma chamada ao método `update_notification()` do formulário vinculado. Neste instante o usuário A visualiza que o usuário B está livre e pode iniciar um bate-papo. Ao mesmo tempo, o Serviço de Contexto do dispositivo A envia uma mensagem de retorno ao dispositivo B indicando o *status* de atualização do contexto que, por sua vez, faz uma chamada ao método `updateContext_answer()` do formulário vinculado ao mesmo. Desta forma, o usuário B pode saber quem acabou de ser notificado da recente alteração do contexto "situação". Ou seja, ambos os dispositivos têm um perfeito controle do processo de atualização de um contexto subscrito.

4.3.4 O Serviço de Mensagem

O Serviço de Mensagens é um serviço de entrega confiável de mensagens sem suporte a desconexão. Ou seja, não há retenção de mensagens para entregas futuras em

caso de falhas na conexão. A mensagem só é entregue caso haja de fato conectividade para tanto. A filosofia deste serviço é prover a funcionalidade de entrega de mensagens da forma mais simples possível, sobre a qual pode-se criar outros serviços de entrega mais sofisticados. Assim como todas as mensagens de rede do *framework*, as mensagens de texto também são protegidas pelo algoritmo de criptografia XXTEA [And03] respeitando a privacidade dos usuários.

Não há limite no tamanho das mensagens a serem enviadas, porém este serviço foi desenhado para o envio de mensagens curtas, deixando o envio de grandes blocos de texto ou documentos grandes para o serviço de dados. Esta funcionalidade permite o desenvolvimento de aplicações como comunicadores pessoais, onde usuários podem armazenar listas de contatos e enviar mensagens de texto quando estiverem disponíveis (e conectáveis).

```
1  package snu;
2
3  public interface iMessageService {
4
5          void setMessageServiceForm(iMessageServiceForm fparam);
6
7          void netSendMessageToContact(int device_pos, String strMessage);
8
9          String getDeviceId();
10
11         String getLastMessage();
12
13         String getLastStatus();
14
15 }
```

Listagem 4.13: Interface iMessageService

Na Listagem 4.13 encontramos a interface do Serviço de Mensagens. Como nos demais serviços, há um método que indica o formulário do usuário responsável pelo processamento das resposta aos comandos de rede (`setMessageServiceForm()`). Uma vez que uma mensagem de texto é recebida, existem métodos para recuperar o identificador de rede do último dispositivo remetente (`getDeviceId()`), recuperar a última mensagem recebida (`getLastMessage()`) e para recuperar o *status* de envio da última mensagem postada (`getLastStatus()`, indicando sucesso ou falha no envio). Para enviar uma mensagem de fato, o *framework* disponibiliza o método `netSendMessageToContact()`, indicando o dispositivo de destino e a mensagem a ser enviada.

```
1  package snu;
2
3  public interface iMessageServiceForm {
4
5          void showMessage(String device_id, String strMessage);
```

```
6
7           void sendStatus(String status);
8
9   }
```

Listagem 4.14: Interface iMessageServiceForm

A Listagem 4.14 exibe a interface do formulário da aplicação do usuário vinculada ao Serviço de Mensagens. Existem apenas dois métodos: `showMessage()` para indicar o recebimento de uma mensagem de um dispositivo e `sendStatus()` para indicar o *status* do envio de uma mensagem de texto (sucesso ou falha).

Implementação

A Figura 4.10 exibe a implementação do Serviço de Mensagem. O Serviço de Mensagem deve implementar a interface `iMessageService` assim como a interface `iService` e, portanto provê todos os métodos nelas declarados. Como ocorre nos demais serviços, existe um referência à interface `iMessageServiceForm` para o processamento das respostas dos comandos de rede.

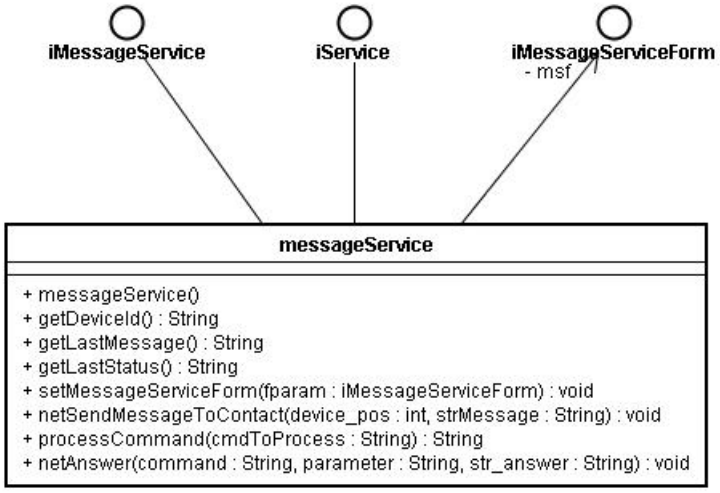


Figura 4.10: Componentes do Serviço de Mensagem

Não há métodos ou propriedades que explicitamente ativem ou desativem a criptografia. O processo é totalmente transparente e atrelado à opção utilizada durante a inicialização do *framework*. Quando o *framework* é iniciado com a criptografia ativa, automaticamente todos as mensagens de rede de todos os serviços são protegidas pelo algoritmo.

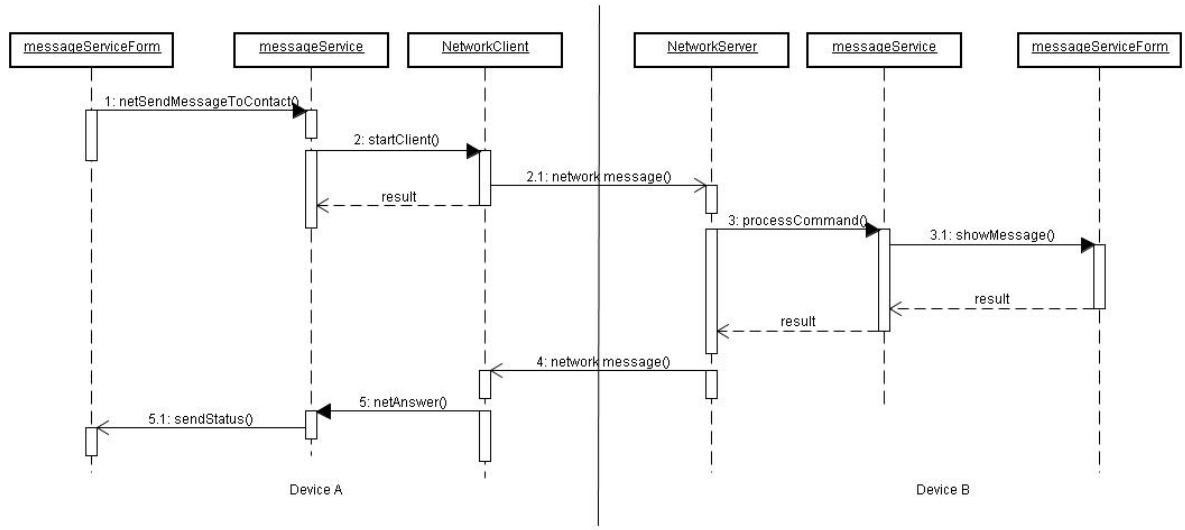


Figura 4.11: Diagrama de seqüência do envio de uma mensagem

A Figura 4.11 mostra o diagrama de seqüência do envio de uma mensagem do dispositivo A para B. O dispositivo A através do método `netSendMessageToContact()` envia uma mensagem de texto ao dispositivo B. O comando é repassado para a camada de rede que o transmite ao dispositivo B. A camada de rede do dispositivo B reconhece o comando de mensagem de texto e o repassa para o Serviço de Mensagens que chama `showMessage()` no seu formulário vinculado. A mensagem é então tratada (muito provavelmente exibida ao usuário em B) e uma resposta automática é gerada. Essa resposta é repassada para a camada de rede do dispositivo A que executa o método `netAnswer()` do Serviço de Mensagens, indicando o *status* de envio recedibo. O serviço então repassa este *status* para o formulário da aplicação vinculado ao Serviço de Mensagens chamando o método `sendStatus()`.

4.4 Resumo

Este capítulo apresentou em detalhes a arquitetura do *framework* SNU e de seus serviços. O SNU é um *framework* para dispositivos móveis em redes *ad-hoc* espontâneas de curto alcance. Se comparado com os diversos trabalhos relacionados na Seção 3, o SNU é o único projeto destinado às redes *ad-hoc* espontâneas. Entende-se por redes espontâneas aquelas formadas ao acaso, de forma aleatória e sem garantias de manutenção de sua topologia.

A arquitetura do SNU é aberta e dividida em três camadas, Camada de Rede,

Camada de Serviços e Camada da Aplicação, com interfaces bem definidas onde cada camada só se comunica com a próxima. Esta arquitetura foi concebida para facilitar o desenvolvimento de novos serviços, assim como, para propiciar a utilização de qualquer tecnologia de rede *ad-hoc*.

Desenvolvido em J2ME, o SNU provê um alto grau de portabilidade. É uma solução capaz de ser executada em celulares convencionais, *Smartphones* ou PDA's, bastando existir uma JVM J2ME disponível com suporte a tecnologia de rede em uso (por exemplo, JSR 82 para *Bluetooth*). Quando comparado aos outros trabalhos visto, o SNU é o único capaz de ser executado sem modificações em praticamente qualquer dispositivo móvel com suporte a JVM J2ME, uma vez que o *MIRES* é uma solução direcionada apenas para celulares com suporte J2ME, enquanto que o *PGWW* e o *Mobile Chedar* são direcionados para PDA's.

Assim como o *ContextPhone* e o *MoCA*, o SNU provê um serviço de contexto, com atualizações dinâmicas e transparentes. Contudo o *MoCA* provê contexto computacionais além de inferir informações de localização, também disponíveis no *ContextPhone*, enquanto que o SNU só disponibiliza contextos da aplicação.

5 Avaliação do *framework*

Para garantir que durante o desenvolvimento do SNU foram contemplados todos os requisitos impostos desde o início do projeto, analisamos individualmente cada requisito e identificamos alguns procedimentos comuns às aplicações do domínio em questão. Realizando diversos experimentos, coletamos dados estatísticos da utilização dos recursos disponíveis, possibilitando assim a identificação de possíveis pontos de melhoria.

Neste capítulo analisamos cada serviço e sua conformidade perante os requisitos impostos, assim como seu comportamento nos variados experimentos realizados. Organizamos nossa análise a partir dos requisitos impostos em 1.1, discorrendo sobre cada serviço individualmente, a fim de demonstrar seu comportamento e seu impacto nas aplicações construídas sobre o *framework*.

5.1 Análise da ausência de controle centralizado

O *framework* do SNU foi elaborado para o uso em redes *ad-hoc* espontâneas, ou seja, redes formadas aleatória e casualmente, comumente composta por participantes desconhecidos e sem a garantia de manutenção de sua topologia. Tomando como base o serviço de contatos (por ser obrigatório) verificamos que as soluções baseadas em serviços transientes [ZII04] [1], assim como as soluções centralizadas para redes *ad-hoc* [2], são incompatíveis com o domínio do problema, porque necessitam que pelo menos N dispositivos sempre façam parte da topologia da rede, o que não pode ser garantido em redes espontâneas. Portanto, soluções que de alguma forma necessitem recuperar informações armazenadas remotamente são inviáveis para o domínio do *framework*.

Desta forma, o serviço de contatos do SNU não utilizou nenhum controle cen-

[1] Serviços transientes são aqueles que não precisam necessariamente residir em um determinado dispositivo porém para serem encontrados necessitam publicar de alguma forma sua localização.

[2] Soluções que centralizam serviços em determinado dispositivo e solucionam eventuais problemas de conexão ou encaminhamento de mensagens [EH00] estão comumente disponíveis em redes de sensores sem fio [CES04].

tralizado de gerenciamento de dispositivos, contatos e grupos, deixando a cargo de cada nó o cuidado de armazenar e gerenciar suas próprias informações. Soluções distribuídas para o gerenciamento de nomes como o DNS [TvS02, CDK00] necessitam que pelo menos N nós (ou suas réplicas) estejam acessíveis para que haja uma resolução de nomes (ou de grupos). Contudo, em redes espontâneas não há garantias de que um dado nó, responsável por armazenar um subconjunto de nomes ou grupos, esteja sempre disponível. A abordagem escolhida simplifica o processo de gerenciamento. Porém, obriga que cada nó tenha o conhecimento prévio dos demais para que haja compartilhamento. Uma vez em que se tenham grupos compartilhados, estes são facilmente importados, o que ameniza a ausência de um diretório central de contatos e grupos.

O serviço de compartilhamento de dados funciona de forma semelhante, armazenando as informações no próprio nó que as gerou. Soluções de armazenamento distribuído como AFS [TvS02] e Coda [Bra98] são soluções infra-estruturadas baseadas em redes confiáveis e, portanto, incompatíveis com o domínio do problema. A cada requisição de uma informação compartilhada no SNU, esta é transmitida a partir do próprio nó e cópias de uma mesma informação em dispositivos diferentes se tornam informações gerenciadas por cada dispositivo detentor da cópia, uma vez que não há um serviço de controle de versões. Contudo, o *framework* disponibiliza funções *hash* para facilitar o desenvolvimento de aplicações que necessitem de um controle de versão sobre itens de dados compartilhados. Uma desvantagem desta abordagem é que como não há um controle centralizado, é imprescindível que o dispositivo esteja acessível para que haja transferência do recurso compartilhado.

O serviço de contexto foi desenvolvido da mesma forma, tendo suas informações armazenadas no nó que as criou e que mantêm seu valor corrente. Sendo assim, cada nó é responsável por propagar as mudanças de valores dos seus contextos para os dispositivos interessados. Esta abordagem simplifica o processo de gerenciamento porém, algumas consultas se tornam difíceis quando não há um serviço centralizado, como aquela existente no *MoCA* [SER+04], onde usuários podem consultar informações de contexto de qualquer dispositivo, inclusive os não cadastrados ou fora de alcance.

5.2 Utilizando poucos recursos

Uma das maiores preocupações no desenvolvimento para dispositivos móveis é a utilização racional dos recursos disponíveis [IF03, Isl04]. Apesar dos constantes avanços tecnológicos, tais dispositivos ainda possuem grandes restrições de processamento, armazenamento, autonomia e interface com usuário. De nada adiantaria desenvolver uma solução que drenasse toda a energia disponível em poucos minutos ou que necessitasse de muito espaço, impossibilitando seu uso concomitante com o armazenamento dos populares arquivos mp3 e fotos.

5.2.1 Serviço de contatos

É sabido que as comunicações em rede sem fio (*link* rádio) são extremamente custosas no tocante ao consumo de energia [IF03]. Desta forma, optamos por não estabelecer conexões permanentes (com constantes trocas de mensagens) e, quando possível, enviamos o máximo de informações por vez. Desta forma, evita-se o uso excessivo da rede, reduzindo assim o consumo de energia.

A função de transferência de grupos de contatos compartilhados pelo serviço de contatos é feita em uma só solicitação. No momento da requisição, o *framework* faz a solicitação enviando a identificação do contato solicitante. No outro lado, ao receber esta requisição, é feita uma validação de permissão para que em seguida sejam enviadas todas as informações dos grupos compartilhados disponíveis. Ou seja, a operação de compartilhamento de grupos necessita de apenas duas mensagens para ser concluída.

5.2.2 Serviço de dados

No tocante ao serviço de dados, as coletâneas são sempre transferidas até uma determinada profundidade (configurável) e jamais são transferidos os itens de dados do tipo arquivos em uma mesma solicitação. Já que não há como prever o tempo necessário para a transferência de itens de dados arquivos pertencente a uma coleção compartilhada, a solução adotada preserva a bateria e a memória de massa, assim como garante que mesmo longas e carregadas coleções possam ser transferidas, porém fracionadamente. Itens de dados *string* são sempre enviados por completo evitando uma segunda solicitação

de envio de seu conteúdo. Itens de dados arquivos são transferidos utilizando-se pequenos *buffers*, lendo e gravando diretamente na memória de massa, reduzindo a alocação de memória da JVM e possibilitando a transferência de arquivos de qualquer tamanho.

5.2.3 Serviço de contexto

Quando há alteração em algum contexto que possua contatos subscritos, o *framework* envia uma mensagem ao contato em questão informando a alteração. Manter uma conexão contínua com todos os contatos subscritos seria por demais custoso e, na medida em que o número de contatos e contextos fosse crescendo, essa solução se tornaria cada vez menos eficiente.

É importante observar uma peculiaridade do serviço de contexto: a qualquer instante, qualquer dispositivo com permissão pode se subscrever em um dado contexto do aplicativo, porém tais subscrições não são persistentes. Pela mesma razão em que não há um controle de versão fornecido pelo *framework*, não há persistência nas informações de subscrição, uma vez que não há garantias de que o contato subscrevente fará parte da topologia da rede novamente.

5.3 Independência da tecnologia de rede

O rápido avanço das tecnologias de rede torna imperativo o desenvolvimento de projetos modulares que facilitem sua adaptação às novas tendências. Projetos que geram um elevado custo de adaptação às novas tecnologias tendem ao fracasso e, por esta razão, o SNU teve como requisito de implementação a independência da tecnologia de rede em uso.

A adoção do *Bluetooth* [Met99] como tecnologia de rede se deu pela própria natureza *ad-hoc* da mesma, assim como pela sua vasta disponibilidade em dispositivo móveis. Contudo, é perfeitamente possível utilizar outras tecnologias de rede como 802.11 em modo *ad-hoc* [Ass03]. Para facilitar a permuta entre as várias tecnologias existentes, o framework foi concebido em camadas, utilizando-se o padrão de fábrica de objetos (de rede) [GHJV94].

A fábrica de objetos é responsável por criar objetos cliente e servidor que

possuam interfaces bem definidas com o restante dos serviços do framework. Desta forma, para utilizarmos qualquer outra tecnologia de rede é necessário apenas o desenvolvimento das classes cliente e servidor implementando as interfaces existentes. Cada serviço tem por responsabilidade inicial registrar suas operações de rede na camada existente, através do objeto servidor. Desta forma, quando o objeto servidor recebe um comando pela rede, saberá qual serviço do *framework* é responsável por processá-lo.

É bom observar que a atual implementação do projeto não possibilita o uso de duas ou mais tecnologias de rede simultaneamente, apesar de possibilitar a implementação e o uso individual de qualquer uma delas, provendo-se assim independência da tecnologia de rede.

5.4 Portabilidade

Para que o SNU possa ser utilizado no maior número possível de dispositivos móveis, o projeto teve como requisito a portabilidade da solução. Atualmente a concorrência do setor de dispositivos móveis fez nascer uma grande variedade de ambientes de execução, muitos deles completamente proprietários e fechados, não possibilitando a utilização de soluções como as apresentadas pelo SNU.

A tecnologia Java para dispositivos móveis, J2ME *Java 2 Micro Edition*, é um subconjunto da especificação Java, amplamente disseminado e disponível na maioria dos dispositivos. E a forma encontrada por diversos fabricantes para permitir a instalação e execução de aplicativos diferentes daqueles disponibilizados por eles. A especificação J2ME é direcionada para o maior número possível de dispositivos móveis e para tanto foi construída sobre a premissa da portabilidade e facilidade de adaptação de sua JVM aos diversos equipamentos disponíveis no mercado.

Contudo, para permitir grande portabilidade, o padrão J2ME possui várias limitações quanto ao gerenciamento de sua JVM e quanto aos pacotes disponíveis em sua especificação padrão, dificultando em parte o desenvolvimento de aplicações. Outra opção analisada foi o *SuperWaba* que, apesar de possuir de um amplo conjunto de pacotes com recursos superiores aos encontrados em J2ME, se encontra disponível apenas para *Smartphones* e PDAs. Também consideramos o uso da linguagem C++. Porém, a grande maioria dos telefones celulares executam sistemas operacionais proprietários e não

disponibilizam interfaces de instalação e gerenciamento de aplicações de terceiros (apesar de disponibilizarem JVM J2ME).

Desta forma, mesmo a um custo de desenvolvimento superior ao de soluções como o *SuperWaba*, o SNU adotou o J2ME como ambiente de desenvolvimento. O framework foi projetado para dispositivos compatíveis com as especificações *Connected Limited Device Configuration* (CLDC) 1.1 [SM03] e *Mobile Information Device Profile* (MIDP) 2.0 [Pro06], facilmente encontradas no mercado.

As especificações CLDC 1.1 e MIDP 2.0 definem um conjunto básico de recursos disponibilizados pela JVM, deixando a cargo de bibliotecas adicionais, especificadas pelas JSRs, *Java Specification Requests*, a implementação de recursos avançados como multimídia, redes e outros. O padrão J2ME não obriga os fabricantes a disponibilizarem todas as JSRs juntamente com seus dispositivos, porém a grande maioria o faz como meio de tornar seus produtos poderosos e versáteis. Para o desenvolvimento deste framework foi necessária a inclusão das JSRs: *PDA Profile for J2ME* (JSR 75 [II04], para o acesso ao sistema de arquivos local) e *Bluetooth/OBEX for J2ME* (JSR 82) [Inc05], ambas amplamente disponíveis.

5.5 Experimentos

Com o objetivo de avaliar a utilização da rede, memória física e de massa e dos ciclos de processamento, conduzimos três experimentos que visam reproduzir os seguintes procedimentos comuns: localização de dispositivos próximos, inicialmente sem sucesso e posteriormente variando-se a quantidade de dispositivos encontrados; transferência de grupos compartilhados, variando-se a quantidade de contatos por grupo; e transferência de itens de dados dos tipos texto e arquivo (variando-se seus tamanhos). Foram tomadas 10 execuções de cada experimento e calculados seus valores mínimos, máximos, a média, o desvio padrão e o intervalo de confiança a 95%.

Todos os experimentos foram realizados com a utilização das ferramentas disponíveis no *Sun Java Wireless Toolkit 2.5 for CLDC*. Utilizamos os valores padrão para o emulador J2ME e coletamos as informações relevantes através dos monitores de rede e de memória, assim como através do *Profiler*. O monitor de rede intercepta e registra todo e qualquer tráfego de rede, independentemente da tecnologia utilizada. No nosso

caso, analisamos apenas o tráfego *Bluetooth*. O monitor de memória coleta informações detalhadas sobre todos os objetos instanciados e sua necessidade de memória durante todo o ciclo de vida dos mesmos. De forma semelhante, o *Profiler* armazena a quantidade de ciclos de processamento (em *byte codes*) utilizados por cada objeto e os exibe de forma unitária ou agrupada em uma estrutura hierárquica (objetos pai). Os resultados obtidos serão analisados individualmente a seguir.

5.5.1 Localização de dispositivos próximos

A localização de dispositivos na vizinhança é um procedimento comum no SNU, repetindo-se em intervalos de tempo regulares cuja duração é definida na instanciação do *framework*. Por esse motivo, analisamos o consumo de recursos requisitado pelo *framework* ao variar-se a quantidade de dispositivos encontrados e a freqüência de execução deste procedimento. Foram medidos o tráfego na rede, o consumo de memória e os ciclos de processamento tomando como base a simples execução de uma aplicação exemplo do SNU que, ao ser iniciada, localiza os dispositivos próximos e registra o encontrado na saída do terminal. Assim que o processo de localização conclui seus trabalhos, a aplicação era encerrada e os seus dados coletados.

Foram analisadas cinco situações distintas: A - nenhum dispositivo foi encontrado; B - um dispositivo foi encontrado; C - dois dispositivos foram encontrados; D - três dispositivos foram encontrados; E - quatro dispositivos foram encontrados; e finalmente F - quando cinco dispositivos foram encontrados. O experimento utilizou os monitores de rede e de memória, assim como o *Profiler*.

Pela própria definição da tecnologia *Bluetooth*, o processo de localização de dispositivos próximos é intrínseco. Portanto, não há troca de informações no nível da aplicação: tudo é feito pelo próprio protocolo utilizando métodos nativos da API. Sendo assim, não houve tráfego de rede reportado pelo monitor de rede, pois este só captura o tráfego gerado pela aplicação, negligenciando as mensagens de controle do próprio protocolo.

No tocante ao consumo de memória, podemos observar na Figura 5.1 que a média do consumo cresceu de forma praticamente linear no decorrer dos experimentos citados. A cada novo dispositivo encontrado, são necessários em média 19KB para o

armazenamento de suas informações. Os níveis mínimos e máximos também seguiram um padrão constante de pouco ou nenhuma variação. Os baixos valores de desvio padrão, vide Tabela 5.1, indicam um comportamento uniforme, o que também se reflete na proximidade dos valores pertencentes ao intervalo de confiança. Esse comportamento uniforme já era esperado, uma vez que para cada novo dispositivo encontrado o *framework* coleta somente suas informações básicas de localização na rede, tornando-as disponível para o gerenciamento de contatos.

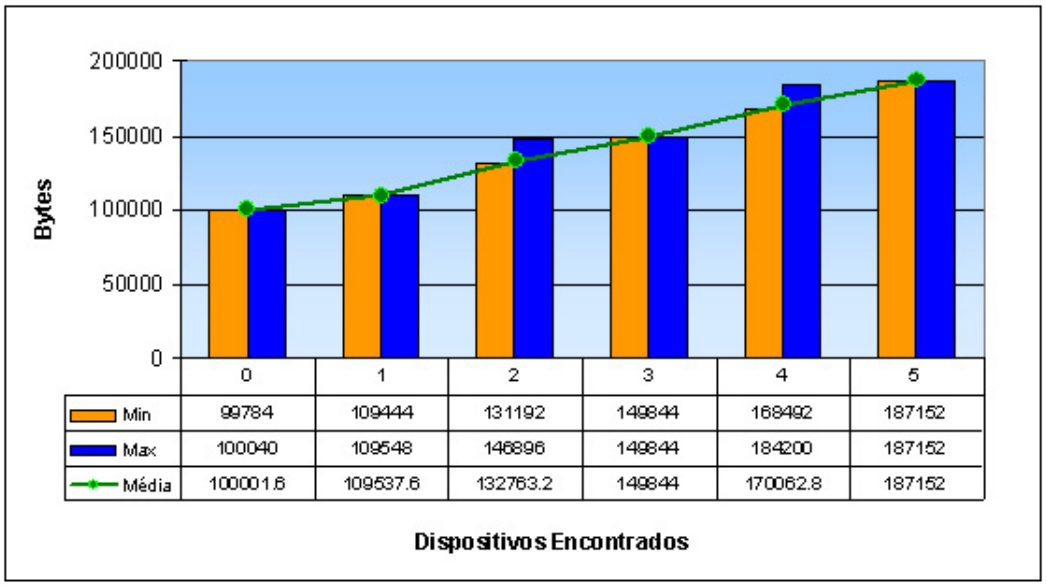


Figura 5.1: Média do consumo de memória na localização de dispositivos

Exper.	Min	Max	Média	Dsv. Padrão	Int. Confiança
A	99784	100040	100001.6	80.95	[99951.42 ; 100051.77]
B	109444	109548	109537.6	32.88	[109517.21 ; 109557.98]
C	131192	146896	132763.2	4965.76	[129685.38 ; 135841.01]
D	149844	149844	149844	0	[149844 ; 149844]
E	168492	184200	170062.8	4967.30	[166984.03 ; 173141.56]
F	187152	187152	187152	0	[187152 ; 187152]

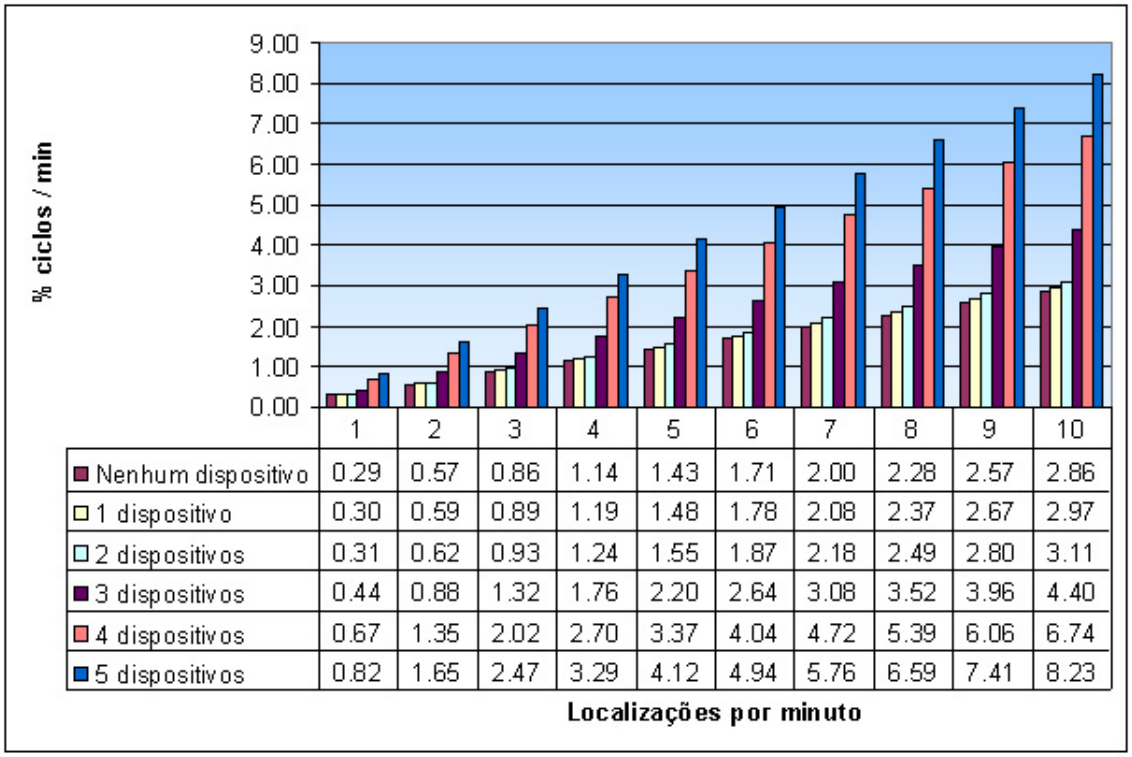
Tabela 5.1: Estatísticas do consumo de memória na localização de dispositivos

Visando quantificar o percentual de ciclos de processamento necessário para o processo de localização de dispositivos próximos, repetimos o experimento anterior variando-se o número de localizações solicitadas por minuto assim como a quantidade de dispositivos localizados. Tomando as leituras dos ciclos de processamento reportados pelo *Profiler*, traçamos o gráfico da Figura 5.2.

Para melhor quantificarmos os resultados deste experimento, utilizamos o recurso de restrição da capacidade de processamento do emulador. Foi determinado que o

emulador deveria se comportar como uma máquina virtual capaz de processar 10.000.000 de *byte codes* por minuto, a menor opção disponível no emulador. Como visamos aferir o percentual de ciclos utilizados pelo *famework* no decorrer deste experimento, é aceitável se considerar a pior capacidade de processamento disponível no emulador, uma vez que os valores percentuais são sempre em relação a capacidade máxima de processamento por unidade de tempo. Desta forma, aferimos a quantidade de ciclos necessária para a localização de nenhum à 5 dispositivos próximos, solicitando que fossem realizadas de 1 a 10 localizações por minuto, construindo assim a Tabela 5.2.

Como esperado, inicialmente o *framework* utiliza poucos ciclos de processamento para realizar uma localização de dispositivos próximos. São necessários apenas 0,29% dos ciclos disponíveis em um minuto de processamento para se determinar a inexistência de dispositivos ao redor e apenas 0,82% para localizar cinco dispositivos. O comportamento se mostra linear à medida em que se aumenta a quantidade de dispositivos. Quando a freqüência de execução do procedimento foi aumentada para dez vezes por minuto (uma vez a cada 6 segundos), utilizou-se apenas 2,86% dos ciclos de processamento disponíveis para se determinar a inexistência de dispositivos ao redor e apenas 8,23% para localizar cinco dispositivos.



	1	2	3	4	5	6	7	8	9	10
Nenhum dispositivo	0.29	0.57	0.86	1.14	1.43	1.71	2.00	2.28	2.57	2.86
1 dispositivo	0.30	0.59	0.89	1.19	1.48	1.78	2.08	2.37	2.67	2.97
2 dispositivos	0.31	0.62	0.93	1.24	1.55	1.87	2.18	2.49	2.80	3.11
3 dispositivos	0.44	0.88	1.32	1.76	2.20	2.64	3.08	3.52	3.96	4.40
4 dispositivos	0.67	1.35	2.02	2.70	3.37	4.04	4.72	5.39	6.06	6.74
5 dispositivos	0.82	1.65	2.47	3.29	4.12	4.94	5.76	6.59	7.41	8.23

Figura 5.2: Ciclos de processamento na localização de dispositivos

Disp. Encontrados	Min	Max	Média	Dsv. Padrão	Int. Confiança
0	0.26	0.30	0.29	0.009	[0.284 ; 0.295]
1	0.24	0.35	0.30	0.041	[0.274 ; 0.325]
2	0.27	0.38	0.31	0.036	[0.287 ; 0.332]
3	0.36	0.55	0.44	0.061	[0.402 ; 0.477]
4	0.44	0.83	0.67	0.125	[0.592 ; 0.747]
5	0.66	1.06	0.82	0.122	[0.743 ; 0.896]

Tabela 5.2: Detalhamento estatístico dos ciclos de processamento na localização de dispositivos da primeira coluna da Figura 5.2 (uma localização por minuto)

Sendo assim, fica evidenciado que o procedimento de localização de dispositivos ao redor não necessita de uma grande fatia de processamento para seu funcionamento. Para a grande maioria das aplicações, é aceitável que o processo de localização se repita 1 à 2 vezes por minuto. Nestas condições, o *framework* precisaria de aproximadamente 0.29% e 1,65% dos ciclos disponíveis para realizar tal procedimento, considerando a localização de zero dispositivo ou cinco dispositivos, respectivamente. Este baixo valor também se reflete em um menor consumo de energia, o que é condizente com o requisito de baixa utilização de recursos. É bom lembrar que outras tecnologias de rede podem utilizar mais ou menos ciclos de processamento para executarem a mesma tarefa atualmente implementada sobre a tecnologia *Bluetooth*.

5.5.2 Transferência de grupos compartilhados

Este experimento visa analisar o comportamento do framework ao transferir grupos de contatos compartilhados. Foram medidos o tráfego na rede, o consumo de memória e os ciclos de processamento tomando como base uma simples aplicação escrita sobre o *framework* SNU. Esta aplicação, ao ser iniciada, localizava de imediato os dispositivos próximos (apenas um neste cenário) e iniciava uma solicitação de transferência de um grupo compartilhado. Foram analisadas três situações distintas quanto ao tamanho do grupo compartilhado: A - grupo compartilhado com apenas um contato; B - grupo compartilhado com dois contatos; e C - grupo compartilhado com três contatos.

Pela própria concepção de utilização dos serviços de rede do framework, todas as informações pertinentes ao grupo compartilhado são transferidas de uma única vez, não importando o tamanho total do mesmo. Sendo assim e como esperado, a cada situação descrita em A, B e C, a única mudança no tráfego de rede foi o crescimento linear, na

razão de um contato (316 bytes), da cadeia de bytes que representa o grupo compartilhado transferido.

Seguindo um comportamento semelhante ao exibido no experimento anterior, a Figura 5.3 mostra um crescimento discreto da utilização de memória a cada situação descrita, acompanhando o aumento na quantidade de contatos transferidos no grupo compartilhado. Os pequenos valores do desvio padrão indicados na Tabela 5.3 nos garantem um comportamento regular e constate do *framework*.

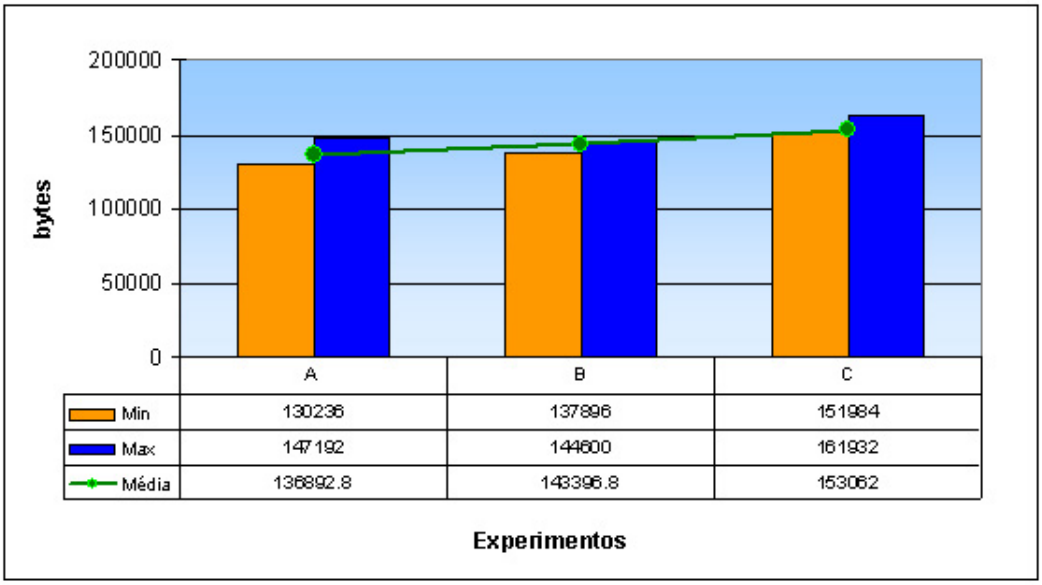


Figura 5.3: Média do consumo de memória na transferência de grupos

Exper.	Min	Max	Média	Dsv. Padrão	Int. Confiança
A	130236	147192	136892.8	4813.81	[133909.16 ; 139876.43]
B	137896	144600	143396.8	2301.71	[141970.18 ; 144823.42]
C	151984	161600	153062	3116.77	[151130.20 ; 154993.79]

Tabela 5.3: Estatísticas do consumo de memória na transferência de grupos

Quanto aos ciclos de processamento utilizados no decorrer do experimento, na Figura 5.4 é possível observar claramente o crescimento na demanda por ciclos de processamento do objeto *contactService* (sempre à direita), uma vez que este é responsável por gerenciar os grupos transferidos. Os demais objetos de rede *btServer* e *btClient* se mantiveram praticamente constantes, apresentando apenas um discreto aumento, refletindo o processamento extra necessário para a transferência de grupos maiores a cada experimento. Novamente temos baixos valores de desvio padrão (Tabela 5.4) denotando uma grande regularidade no experimento.

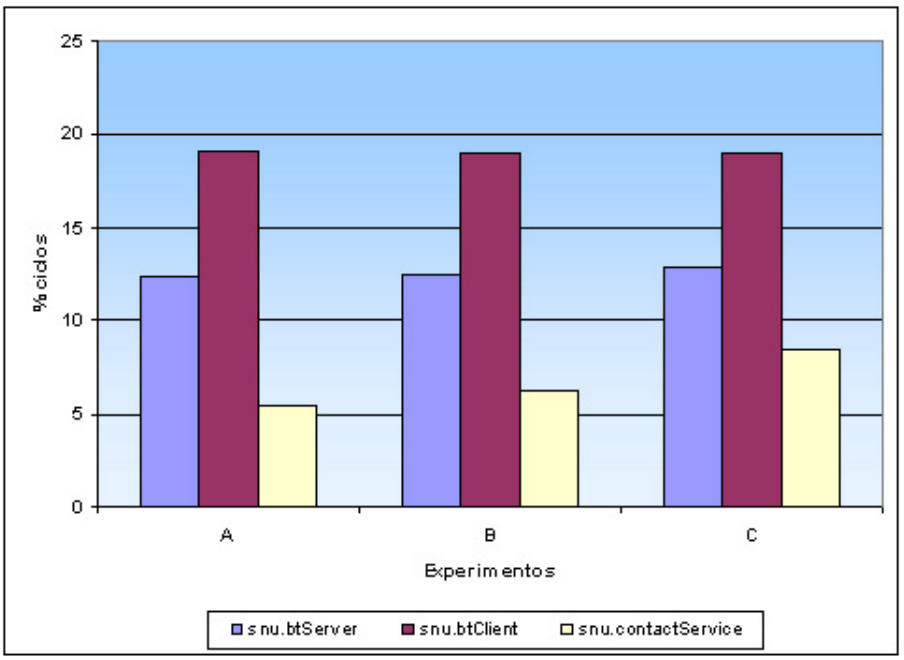


Figura 5.4: Média dos ciclos de processamento na transferência de grupos

Exp.	**Objeto**	**Min**	**Max**	**Média**	**Dsv.Padrão**	**Int.Confiança**
A	snu.btServer	11.1	13.6	12.34	0.83	[11.82 ; 12.85]
	snu.contactService	4.7	6	5.42	0.46	[5.13 ; 5.70]
	snu.btClient	15.8	23.9	19.09	2.66	[17.43 ; 20.74]
B	snu.btServer	10.4	14.4	12.44	1.13	[11.73 ; 13.14]
	snu.contactService	4.3	7.1	6.27	0.83	[5.75 ; 6.78]
	snu.btClient	16.7	23.1	19.03	2.31	[17.59 ; 20.46]
C	snu.btServer	11.1	16.9	12.86	1.57	[11.88 ; 13.83]
	snu.contactService	7.2	9.8	8.43	0.87	[7.88 ; 8.97]
	snu.btClient	18.2	20.2	19.05	0.70	[18.61 ; 19.48]

Tabela 5.4: Estatísticas dos ciclos de processamento na transferência de grupos

Este experimento evidencia que o procedimento de transferência de grupos compartilhados do *framework* não necessita de fatias substanciais de memória além do mínimo necessário para armazenar as informações extras de cada novo grupo compartilhado. Da mesma forma, a demanda por ciclos de processamento se altera de fato apenas no objeto *contactService*, responsável por gerenciar tais informações. Seu discreto aumento na demanda por ciclos de processamento e de memória também se refletem em um menor consumo de energia. Estes resultados, assim como a estabilidade dos demais objetos são condizentes com o requisito de baixa utilização de recursos.

5.5.3 Transferência de itens de dados

Utilizamos este experimento para analisar o comportamento do framework na transferência de um item de dado variando-se o seu tamanho. De forma análoga aos demais experimentos, foram medidos o tráfego na rede, o consumo de memória e os ciclos de processamento. Tomamos como base uma aplicação escrita sobre o *framework* SNU que ao ser iniciada, localizava de imediato os dispositivos próximos (apenas um também neste cenário) e iniciava uma solicitação de procura por itens de dados compartilhados e sua subseqüente transferência. Logo em seguida, a aplicação era finalizada e seus dados coletados.

Inicialmente analisamos três situações onde se alterava apenas o tamanho dos itens de dados transferidos: A - item com 10 bytes; B - item com 15 bytes; e C - item com 20 bytes. Os teste indicaram um comportamento semelhante ao experimento 5.5.2 onde existia um aumento de tráfego de rede e memória linearmente proporcional ao aumento dos objetos transferidos. Também foi mantido o mesmo comportamento quanto à demanda por ciclos de processamento, porém com a utilização do objeto *dataService*.

Em seguida repetimos o mesmo experimento, porém com itens de dados arquivos (100KB, 200KB e 300KB). Foi constatado o mesmo comportamento quanto à demanda por ciclos de processamento, contudo sem variação representativa do consumo de memória. Este comportamento foi previsível, uma vez que o *framework* (por definição da arquitetura) transfere os arquivos utilizando um *buffer* de 10KB, e os armazena diretamente na memória de massa disponível.

Como o comportamento do *framework* foi semelhante ao exibido no experimento anterior, fica evidenciado que o procedimento de transferência de itens de dados de fato se comporta como o procedimento de transferência de grupos compartilhados. Ambos não necessitam de fatias substanciais de memória além do mínimo necessário para armazenar as informações extras de cada novo item compartilhado. Da mesma forma, o discreto aumento na demanda por ciclos de processamento do objeto *dataService*, assim como a estabilidade dos demais objetos são condizentes com o requisito de baixa utilização de recursos.

5.5.4 Conclusões Sobre os Experimentos

Os experimentos descritos procuraram reproduzir procedimentos e operações que consideramos comuns às aplicações escritas sobre o *framework* SNU. Os dados coletados refletem a utilização da rede, memória e processamento durante a execução das diversas aplicações de avaliação. Como esperado, o esforço de desenvolvimento desprendido no intuito de acomodar o requisito de baixa utilização de recursos foi recompensado pelas conclusões reveladas após a análise dos dados coletados em cada experimento realizado. Como um todo, o *framework* manteve um uso discreto dos recursos disponíveis, liberando a maior parte do potencial do dispositivo para as aplicações criadas com seu auxílio.

5.6 Aplicações

Como o objetivo deste trabalho é apresentar um *framework* que auxilie os desenvolvedores nos projetos para dispositivos móveis em redes *ad-hoc* espontâneas de curto alcance, identificamos alguns cenários e especificamos algumas aplicações a fim de avaliar empiricamente se atingimos no todo ou em parte os objetivos propostos. Os mais variados cenários podem ser estudados, desde aplicações de compartilhamento de informações pessoais, como uma agenda de grupo ou uma troca de fotos, até aplicações sofisticadas de auxílio pedagógico como anotações, exercícios ou notas de aula compartilhados diretamente a partir do dispositivo móvel do professor (vide Seção 1.1).

Desta forma, desenvolvemos e analisamos inicialmente duas aplicações colaborativas, *SNU Visit Card* e *SNU In Touch*, e especificamos outras, como o *SNU Data Explorer*, que ainda se encontra em processo de implementação (e portanto não faz parte desta análise). Estas aplicações foram desenvolvidas por estudantes de graduação do curso de Ciência da Computação da Universidade Federal do Maranhão que possuíam conhecimentos básicos sobre a tecnologia J2ME, e tiveram acesso irrestrito a toda documentação e código fonte do *framework*. Também foi disponibilizado um contato constante com o autor deste trabalho, a fim de sanar quaisquer dúvidas que porventura viessem a apresentar.

A seguir descrevemos as funcionalidades de cada aplicação desenvolvida, ap-

resentando algumas telas de sua execução no emulador do SDK da *Sun Microsystems*, as dificuldades encontradas pelo desenvolvedor, assim como os percentuais de código destinados a interface com o usuário, às regras de negócio e ao SNU em si. Para o cálculo deste percentual foi criado um arquivo de palavras reservadas do SNU com todos os nomes de interfaces, classes, métodos e propriedades presentes em seu pacote e, utilizando-se os comandos Unix `grep` e `wc` calculamos o total de linhas de código que faziam referência a alguma dessas palavras reservadas. Além dos teste no emulador, as aplicações também foram testadas nos celulares dos membros do grupo de trabalho deste projeto (modelos Sony Ericsson K750i, K790i e Morotola C30), apresentando o comportamento esperado.

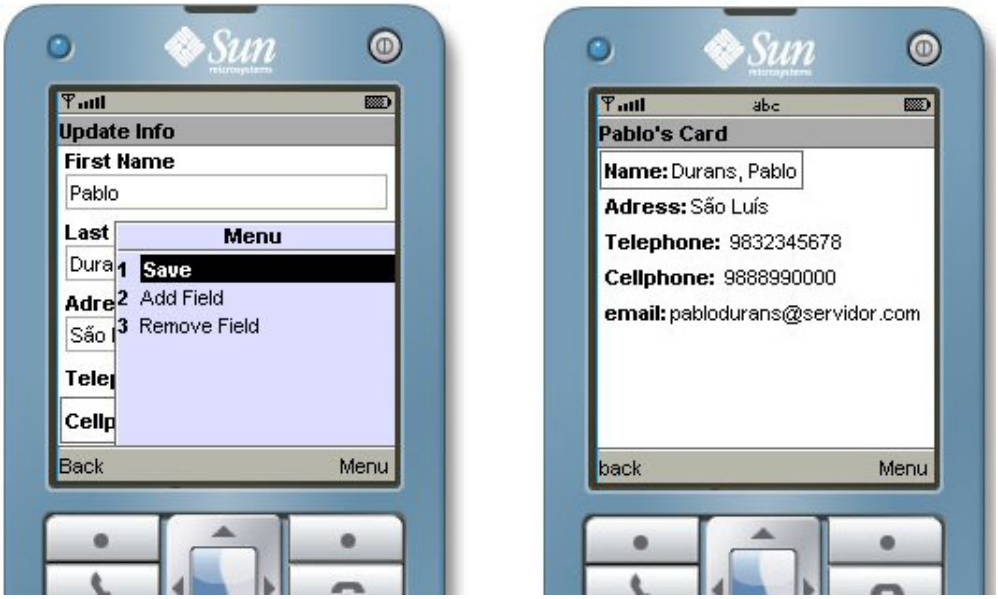
5.6.1 SNU Visit Card

O *SNU Visit Card* é um aplicativo para criação, gerenciamento e troca de cartões de visita via *Bluetooth*. O usuário cria seus cartões de visita (quantos achar necessário) e os compartilha com outros usuários ou grupos. Inicialmente os cartões criados possuem cinco campos pré-definidos: nome, sobrenome, telefone residencial, celular e endereço. Contudo o usuário pode adicionar livremente outros campos aos seus cartões. Cada cartão de visita pode ter permissões de compartilhamento distintas, tornando possível o compartilhamento de cartões apenas com certos usuários ou grupos. A qualquer instante o usuário pode localizar outros usuários ao redor e solicitar a transferência dos seus cartões de visita. No âmbito do SNU, os cartões de visita são representados por itens de dados. Desta forma, a aplicação fez uso do Serviço de Contatos e do Serviço de Dados.

A Figura 5.5 mostra as telas principal (a) e de edição dos cartões de visita (b), enquanto que a Figura 5.6 mostra as opções de edição dos cartões de visita (a) e a visualização dos mesmos (b). O programador levou uma semana, trabalhando entre 3 a 4 horas diárias para desenvolver a aplicação, e não relatou nenhum problema específico durante seu desenvolvimento, considerando o SNU simples e objetivo. O código final possui 1033 linhas, onde 878 foram utilizadas pela interface com o usuário, 81 pelas regras de negócio e 74 pelas funcionalidades do SNU, representando apenas 7,2% do código. Ou seja, a maior parte do esforço de programação foi gasto com a interface do usuário, uma vez que as questões relativas ao gerenciamento (persistência) e compartilhamento (transferência) dos cartões (itens de dados) são tratadas pelo *framework*.

(a) (b)

Figura 5.5: Tela principal (a) e editando um cartão de visita (b)



(a) (b)

Figura 5.6: Opções de edição (a) e visualizando um cartão de visita (b)

5.6.2 SNU In Touch

O *SNU In Touch* é um aplicativo para troca de mensagens de texto via *Bluetooth*. É possível procurar por contatos através de seus dados compartilhados, cadastrá-los organizando-os em grupos, trocar mensagens de texto e receber notificações de sua disponi-

bilidade (livre, ocupado, estudando e etc.). Para adicionar um contato a lista de contatos o usuário deverá enviar um convite ao novo contato solicitando seu consentimento, que por sua vez responderá afirmativamente ou não. O gerenciamento dos contatos e listas é feito pelo Serviço de Contatos e as trocas de mensagens são feitas pelo Serviço de Mensagens do SNU. O usuário pode cadastrar seus interesses pessoais criando chaves de pesquisa por novos contatos. O compartilhamento de dados é gerenciado pelo Serviço de Dados e pode ser utilizado tanto para compartilhar dados simples como os interesses pessoais, como arquivos locais (fotos, toques, vídeos e etc.). Além da transferência dos dados, os usuários podem verificar possíveis alterações sem a necessidade de transferi-los novamente, utilizando as funções *hash* e MD5. Usuários possuem ainda um indicador de disponibilidade (ocupado, livre, estudando...) representado por um contexto (Serviço de Contexto), responsável por manter atualizado o indicador em todos os contatos da lista.

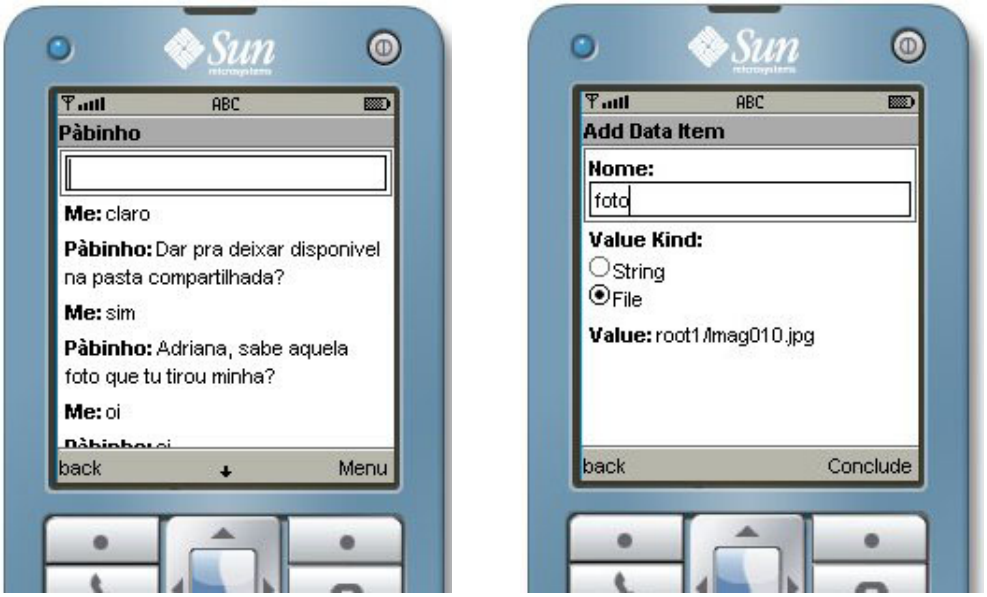
(a) (b)

Figura 5.7: Tela principal (a) e editando interesses pessoais (b)

A Figura 5.7 mostra as telas principal (a) e de edição dos interesses pessoais (b). A Figura 5.8 mostra as telas de localização de contatos ao redor (a) e a tela principal no momento em que uma mensagem foi enviada por um contato cadastrado (b). A Figura 5.9 mostra as telas de conversação (a) e de compartilhamento de um arquivo, uma imagem no caso (b). Dada a maior complexidade do projeto, o programador precisou de cerca de um mês, trabalhando entre 3 a 4 horas diárias para desenvolver a aplicação, e não relatou nenhum problema específico durante seu desenvolvimento, contudo, atribuiu o

(a) (b)

Figura 5.8: Localizando contatos ao redor (a) e recebendo uma mensagem (b)



(a) (b)

Figura 5.9: Conversação (a) e compartilhando um arquivo (b)

tempo longo de desenvolvimento a sua pouca familiaridade com J2ME. O código final possui 2646 linhas, onde 2090 foram utilizadas pela interface com o usuário, 303 pelas regras de negócio e 253 pelas funcionalidades do SNU, que representaram apenas 9,6% do código. Novamente observamos o grande esforço de programação gasto na interface com o usuário, uma vez que há poucas regras de negócio e que todo o gerenciamento de contatos, disponibilidade (contexto) e transferência de mensagens é feito pelo *framework*.

5.6.3 Outras Aplicações

Dentre as aplicações atualmente em desenvolvimento podemos ainda citar o *SNU Data Explorer*. Visando o compartilhamento de informações, através dos objetos itens de dados do *framework* (Serviço de Dados), o *SNU Data Explorer* foi idealizado para gerenciar e compartilhar informações diversas utilizando a tradicional filosofia dos sistemas de arquivos, ou seja, organizando as informações em pastas. Os usuários poderiam então criar e compartilhar pasta com seus contatos, que a qualquer instante, poderiam procurar por novos dados compartilhados e solicitar sua transferência.

Por fim, temos os jogos de cartas e tabuleiros que também podem fazer uso do *framework* SNU. Por indicação de um dos desenvolvedores e atualmente em fase de análise, o popular Can-can (jogo de cartas) deverá ser o primeiro jogo construído sobre o SNU. As possibilidades são inúmeras e, com a adição de novos serviços, novos cenários serão descobertos, originando aplicações mais sofisticadas capazes de tornar o uso dos dispositivos móveis mais prático e divertido.

5.6.4 Conclusão

Os dois projetos citados apresentaram menos de 10% de seu código final dedicado a utilização das funcionalidades do SNU, demonstrando que a maior parte de seu código foi dedicada a interface com o usuário e as regras de negócio da aplicação. Desta forma, pode-se concluir que o SNU apresenta uma interface de serviços prática e objetiva, disponibilizando um leque de funcionalidades suficiente para sua utilização sem a necessidade de um considerável esforço de programação. Também foi notória a ausência de queixas a sua interface, o que nos leva a crer que o SNU atendeu as necessidades encontradas.

6 Conclusão e Trabalhos Futuros

A popularização das tecnologias de rede de curto alcance, como Bluetooth, possibilitou o surgimento de uma nova forma de interação. Usuários podem agora se reunir de forma rápida e ocasional, dando origem a um tipo de rede de computadores dita espontânea. Nesse modelo de interação, usuários se reúnem em alguma tarefa colaborativa, ocasional e quase sempre de curta duração, trocando informações através de redes *ad-hoc*, sem a necessidade de infra-estrutura e sem custo financeiro de comunicação.

Na tentativa de expandir a utilidade destes dispositivos e assim prover um maior grau de interação entre os usuários, este trabalho apresentou o SNU - *Spontaneous Network Utility*, um *framework* cujo objetivo é auxiliar o desenvolvimento de aplicações para dispositivos móveis em redes *ad-hoc* espontâneas de curto alcance. O SNU possui uma arquitetura aberta, organizada em camadas e expansível. Ele provê diversos serviços, como gerenciamento de contatos, compartilhamento de dados e informações de contexto e troca de mensagens, não requerendo uma infra-estrutura de comunicação nos locais de sua utilização dado que sua concepção é voltada à redes ad-hoc, um aspecto relativamente pouco explorado em trabalhos de pesquisa relacionados.

Foram definidos quatro requisitos básicos de operação para este *framework*. Primeiramente foi definido que não deveria existir um controle centralizado, ou seja, cada dispositivo deveria gerenciar seus dados e serviços sem depender de um controle externo. Em seguida definiu-se que o *framework* deveria consumir poucos recursos, tornando sua utilização viável em um grande espectro de dispositivos móveis. Por fim, foi determinado que dever-se-ia prover independência da tecnologia de rede, além de apresentar uma solução com grande portabilidade.

Foram apresentadas as soluções para cada requisito e os testes e experimentos aplicados tornaram possível avaliar sua eficiência e eficácia. A abordagem adotada garantiu o desenvolvimento de soluções baseadas nas interações ponto-a-ponto, sem controle centralizado ou infra-estrutura. Os experimentos mediram o consumo de memória e ciclos de processamento de alguns cenários cotidianos de utilização do *framework*, constatando-

se a pequena utilização de recursos. Sua arquitetura em camadas e serviços garantiu a independência da tecnologia de rede e a utilização de J2ME ratificou sua portabilidade.

As aplicações desenvolvidas sobre o *framework* proposto ajudaram a avaliar sua utilidade fora dos limites do laboratório. Os desenvolvedores envolvidos neste projeto relataram facilidade no manuseio de suas classes e o código final das aplicações de avaliação apresentou grande percentual dedicado à interface com o usuário, uma vez que todo o trabalho de comunicação, armazenamento e compartilhamento é realizado pelo *framework*. Observa-se ainda que o SNU disponibiliza mecanismos de proteção de objetos compartilhados (como dados e listas de contato) através do uso de listas de controle de acesso e uso opcional de criptografia simétrica na comunicação.

A partir destes resultados, pode-se concluir que a abordagem e o desenvolvimento deste trabalho de fato ajudaram a solucionar alguns dos problemas enfrentados pelos desenvolvedores no tocante às aplicações móveis em redes *ad-hoc* espontâneas de curto alcance. A utilização de J2ME garantiu sua viabilidade em celulares, *Smartphones* e PDA's, tornando-o uma solução plausível para os problemas citados.

6.1 Trabalhos Futuros

Ao longo do desenvolvimento deste trabalho surgiram diversas idéias que poderiam ser exploradas em trabalhos subseqüentes, bem como foram identificados aspectos do *framework* SNU que necessitam ser aprofundados, dentre os quais ressaltamos:

- A implementação atual deste projeto contempla apenas o *Bluetooth* como tecnologia de rede de curto alcance. Contudo, outras tecnologias como a 802.11 em modo *ad-hoc* poderiam ser utilizadas. Desenvolver outras camadas de rede integrando outras tecnologias aumentaria a portabilidade e usabilidade deste trabalho.

- O sistema de segurança atualmente implementado não contempla a criptografia dos arquivos compartilhados assim como não possui nenhum método de autenticação. Aprimorar este sistema criptografando os arquivos antes de transmiti-los e utilizando um serviço de autenticação, com uso de criptografia com chaves públicas e privadas, aumentaria consideravelmente a segurança da solução.

- Observou-se a possibilidade de se explorar outras abordagens para o modelo de

segurança do *framework*, como o uso de redes de confiança [dRBPJK06].

- A implementação de um mecanismo para controle de versões de itens de dados é um possível aprimoramento do serviço de dados que poderia ser útil, por exemplo, para grupos que trabalham cooperativamente na edição de documentos.

- A implementação de um serviço de dados que contemple não somente a informação, mas também a sua forma, como uma linguagem de marcação.

- O desenvolvimento de novos serviços como o compartilhamento de *streams* de áudio e vídeo expandiria os horizontes deste projeto no sentido das aplicações de monitoramento e dos jogos colaborativos.

- A integração do SNU com projetos voltados para redes sem fio infra-estruturadas, como o *MoCA*, possibilitaria expandir os serviços disponíveis às aplicações, caso os dispositivos estejam em um ambiente que possua uma infra-estrutura de acesso à rede.

- A integração com serviços de localização tornaria possível o desenvolvimento de aplicações vinculadas a determinadas áreas geográficas, como um mural (ou um quadro de recados) eletrônico disponível em cada sala de aula de uma escola.

- A inclusão de ciência de contextos computacionais, como largura de banda disponível e carga de processamento, possibilitaria o desenvolvimento de aplicações adaptativas, capazes de tomar decisões como aumentar a taxa de compressão de um *stream* de áudio ao observar que há disponibilidade de recursos.

- A inclusão de um acesso direto a informação de contexto atual, sem a necessidade de subscrição.

- A utilização de valores contínuos para contextos.

- O SNU poderia ser explorado para o desenvolvimento de novas aplicações em diversos domínios como o educacional (para disseminação de notas de aula e listas de exercício, por exemplo) ou corporativo (para apoio a reuniões e agenda de compromissos, entre outras aplicações).

- Avaliar a utilização dos recursos quando sucessivas tentativas de transferência de dados na rede são fracassadas.

- Prover interação entre dispositvos próximos que não estejam visíveis através do encaminhamento de mensagens (vide *Scatternet*).

- Informações de contexto distribuídos, onde cada dispositivo armazenaria as informações de contextos dos outros formando uma rede de serviços de contexto.

Referências Bibliográficas

[AGIS00] Frank Adelstein, Sandeep K.S. Gupta, Golden G. Richard III, and Loren Schiwiebert. *Fundamentals of Mobile and Pervasive Computing.* Addison-Wesley, 2000. 3rd edition.

[And03] Vikram Reddy Andem. A cryptanalysis of the tiny encryption algorithm. Master's thesis, The University of Alabama, 2003.

[Ass] Infrared Data Association. Irda the secure wireless link. http://www.irda.org.

[Ass03] IEEE Standards Association. 802-11:1999/amd 1:2000(e) Standard Specification, June 2003. http://standards.ieee.org.

[Bea05] Russel Beale. Supporting social interaction with smart phones. *PERVASIVE Computing IEEE CS and IEEE ComSoc*, 2005.

[Bra98] Peter J. Braam. The coda distributed file system. *Linux Journal N.50*, June 1998.

[CDK00] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: concepts and design.* Addison-Wesley, 2000. 3rd edition.

[CES04] David Culler, Deborah Estrin, and Mani Srivastava. Overview of sensor networks. *IEEE Computer Society, Computer*, pages 41–49, Aug 2004.

[dRBPJK06] Jose de Ribamar Braga Pinheiro Junior and Fabio Kon. Representando opinioes em cadeias de confianca spki/sdsi. *Simposio Brasileiro de Seguranca da Informacao e de Sistemas Computacionais - SBC*, Sep 2006.

[EH00] C. Elliott and B. Heile. Self-organizing, self-healing wireless networks. *IEEE Int'l Conf. on Personal Wireless Communications*, pages 355–362, 2000.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley Professional, Oct 1994.

[Gry] Eugene A. Gryazin. Service discovery in bluetooth. Group for Robotics and Virtual Reality, Department of Computer Science, Helsinki University of Technology.

[HMNS03] Uwe Hansmann, Lothar Merk, Martin S. Nicklous, and Thomas Stober. *Pervasive Computing: The Mobile World*. Springer, 2003. 2nd edition.

[IF03] Nayeem Islam and M. Farad. Thinking objectively: Towards ubiquitious acceptance of ubiquitous computing. *ACM Communications*, Feb 2003.

[II04] IBM and PalmSource Inc. Jsr 75: Pda optional packages for the j2metm platform, Jun 2004. Specification Version 2.1.

[Inc05] Motorola Inc. Java api for bluetooth wireless technology (jsr 82), Sep 2005. Specification Version 1.1.

[Isl04] Nayeem Islam. From smart to autonomous phones. *Pervasive Computing, IEEE CS and IEEE ComSoc*, Jul 2004.

[KPB+04] Marc-Olivier Killijian, David Powell, Michael Banare, Paul Couderc, and Yves Roudier. Collaborative backup for dependable mobile applications. *Middleware for Pervasive and Ad-Hoc Computing*, pages 146–149, 2004. Middleware 2004 Companion.

[Lab] California Software Laboratories. Programming with infrared sockets whitepaper. http://www.cswl.com/whiteppr/white/infrared.html. The Outsourced Product Development Partner.

[LLZW04] Qing Li, Xiang Li, Jian Zhai, and Liu Wenyin. Mires - an information exchange system for mobile phones. *ACM Symposium on Applied Computing*, 2004. City University of Hong Kong.

[Met99] Riku Mettala. Bluetooth protocol architecture. Technical report, The Official Bluetooth Membership Site, Aug 1999. Doc. 1.C.120/1.0.

[NMMJ05] Kotilainen N., Weber M., Vapa M., and Vuori J. Mobile chedar - a peer-to-peer middleware for mobile devices. *PerCom 2005 Workshops - IEEE*, 2005.

[PGGH03] Thomas Plagemann, Vera Goebel, Carsten Griwodz, and Pal Halvorsen. Towards middleware services for mobile ad-hoc network applications. *Proc. The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, 2003.

[Pro06] Java Community Process. Mobile information device profile for javatm 2 micro edition, May 2006. Specification Version 2.1.

[RIS05] Manuel Roman, Nayeem Islam, and Shahid Shoaib. A wireless web for creating and sharing personal content through handsets. *PERVASIVE Computing IEEE CS and IEEE ComSoc*, pages 67–73, 2005. DoCoMo USA Labs, San Jose, CA.

[Riv92] Ronald Rivest. The md5 message-digest algorithm. Technical report, MIT Laboratory for Computer Science, 1992. RFC1321.

[ROPT05] Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen. Contextphone: A prototyping platform for context-aware mobile applications. *PERVASIVE Computing IEEE CS and IEEE ComSoc*, pages 51–59, 2005. University of Helsinki and Helsinki Institute for Information Technology.

[SER+04] Vagner Sacramento, Markus Endler, Hana K. Rubinsztejn, Luciana S. Lima, Kleder GonÃ§alves, Fernando N. Nascimento, and Giulliano A. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems*, Oct 2004.

[SM03] Inc. Sun Microsystems. Connected limited device configuration, May 2003. Specification Version 1.1.

[SS04] Bill N. Schilit and Uttam Sengupta. Device ensembles. *IEEE Computer Society, Computer*, pages 56–64, Dec 2004.

[Sta02] William Stallings. *Wireless communications and networks.* Prentice-Hall, 2002.

[TvS02] Andrew S. Tanenbaum and Maaten van Steen. *Distributed Systems: principles and paradigms.* Prentice-Hall, 2002.

[WPD+02] Roy Want, Trevor Pering, Gunner Danneels, Muthu Kumar, Murali Sundar, and John Light. The personal server: Changing the way we think about ubiquitous computing. *Proc. 4th Intl Conf. Ubiquitous Computing*, June 2002. Intel Research, Santa Clara, CA.

[Yua02] Michael J Yuan. Access web services from wireless devices. http://www.javaworld.com, Aug 2002. Wireless Java Session, Java World, Fueling Innovation.

[ZII04] Dong Zhou, Nayeem Islam, and Ali Ismael. Flexible on-device service object replication with replets. *ACM WWW*, pages 131–142, 2004. DoCoMo USA Labs, San Jose, CA.