UNIVERSIDADE FEDERAL DE SANTA MARIA

CENTRO DE TECNOLOGIA
CURSO CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE GRADUAÇÃO

Serviço de Autenticação baseado em Credenciais para Espaços Pervasivos

Fábio Lorenzi da Silva

Santa Maria, RS, Brasil

2006

SERVIÇO DE AUTENTICAÇÃO BASEADO EM CREDENCIAIS PARA ESPAÇOS PERVASIVOS

POR

Fábio Lorenzi da Silva

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria, como requisito parcial para a obtenção do grau de **Bacharel em Ciência da Computação**.

Profa. IARA AUGUSTIN
Orientador

Prof. JOÃO CARLOS DAMASCENO LIMA
Co-orientador

Trabalho de Graduação nº 220

Santa Maria, RS, Brasil

2006

Universidade Federal de Santa Maria

Centro de Tecnologia

Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada, aprova o Trabalho de Graduação

SERVIÇO DE AUTENTICAÇÃO BASEADO EM CREDENCIAIS PARA ESPAÇOS PERVASIVOS

elaborado por

Fábio Lorenzi da Silva

como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

COMISSÃO EXAMINADORA:

_____

Profa. Dra. Iara Augustin

(Orientador)

_____

Prof. Bel. Antônio Marcos de Oliveira Candia

_____

Msc. Celio Trois

Agradecimentos

É com imensa alegria e satisfação que após finalizado o trabalho escrevo esta parte, para assim agradecer as pessoas que, direta ou indiretamente, participaram e ajudaram durante a realização do presente trabalho, e durante toda a caminhada trilhada até a formatura.

Em especial, agradeço imensamente a minha mãe, Fátima, por não medir esforços para eu chegar até aqui, e ainda pelo apoio incondicional e pela palavra amiga, sincera e de estímulo nos momentos que parecia não ter força e capacidade para continuar. Sei que muitas vezes deixou de lado seus planos e objetivos para propiciar e atender aos meus, e sou eternamente grato por isso. Além dela agradeço ao meu pai, Alberi, por ter me ensinado muitas coisas que guardarei para o resto da minha vida, e também por sempre ter me incentivado a estudar e a ser um Homem com caráter e dignidade. Mesmo não estando presente fisicamente comigo sei que está muito próximo, e que me acompanha constantemente na minha caminhada. A saudade é gigantesca, mas diminuída pela certeza que um dia nos reencontraremos. Espero poder educar e acompanhar meus filhos assim como vocês fizeram a mim, pois sem isso tudo não teria chegado até aqui.

A minha namorada, Daniane, pela compreensão em todos os momentos, principalmente naqueles em que eu estava estressado com algumas coisas referentes a faculdade e a minha vida particular. Agradeço ainda pelo companheirismo, conselhos e pela força que sempre me deu para que eu pudesse transpor os obstáculos que foram surgindo. Certamente a sua ajuda foi muito importante para que tudo isso esteja acontecendo.

Agradeço ainda ao meu irmão, Fabrício, pela amizade, carinho, afeto e por ter me presenteado, junto com a Catiane, com o sobrinho e afilhado mais lindo e especial que eu podia ter, o Dérik. Ainda, agradeço ao Marino pela amizade e companheirismo. Sempre disposto a me ajudar, e até mesmo se sobrecarregando nas atividades suprindo as minhas ausências que muitas vezes se deram por falta de tempo.

Em especial, agradeço a minha orientadora, Iara Augustin, pelo apoio e eterna disposição em auxiliar e a me orientar desde o primeiro momento em que começamos a trabalhar juntos lá pelo terceiro semestre. Agradeço ainda a oportunidade recebida no Mestrado e tenho certeza que irei desempenhar um bom trabalho com a sua orientação. Agradeço também ao Caio, meu co-orientador, pela grande disponibilidade, ajuda, conselhos durante a realização do presente trabalho, durante a graduação e nas atividades no gMob. Certamente vocês foram muito importantes para a conclusão dessa etapa, e espero que futuros trabalhos sejam tão bem realizados quanto os já feitos.

Aos amigos de Restinga Sêca, minha querida cidade, pela amizade, companheirismo, e certeza que sempre posso contar a ajuda de vocês. Ainda, aos colegas de graduação, e do gMob, que se tornaram grandes amigos. Destes ficam as lembranças de muitos acontecimentos, ajuda, e a certeza que nos encontraremos nas próximas etapas de nossas vidas. Desejo muito sucesso a todos.

Agradeço também a meus avós, tios, primos e familiares por todo o carinho e estímulo dispensado a mim nessa caminhada. Por último e não menos importante, agradeço a Deus por tudo que tem me propiciado na vida. Sou grato pelas experiências vividas, pelo amparo e pelos familiares e amigos que me rodeiam e que participam da minha vida.

Sumário

Lista de Figuras

RESUMO

Trabalho de Graduação
Ciência da Computação
Universidade Federal de Santa Maria

SERVIÇO DE AUTENTICAÇÃO BASEADO EM CREDENCIAIS PARA ESPAÇOS PERVASIVOS

AUTOR: FÁBIO LORENZI DA SILVA
ORIENTADOR: Profa. Dra. IARA AUGUSTIN
CO-ORIENTADOR: Prof. Msc. JOÃO CARLOS DAMASCENO LIMA

A computação pervasiva surge como uma nova área da computação criando uma visão de mobilidade global, onde a computação está totalmente integrada ao ambiente físico do usuário, objetivando auxiliá-lo em suas tarefas diárias. Este novo cenário da computação ainda propõe o deslocamento da visão centrada nos processos computacionais para a visão centrada no usuário e suas atividades, onde quem deve ser reconhecido pelo sistema deve ser o usuário e não os equipamentos que ele utiliza, pois ele pode utilizar vários dispositivos durante a execução de uma mesma aplicação. Nesse sentido, a contribuição deste trabalho é o desenvolvimento de um serviço de autenticação baseado em credenciais que mantenha a mobilidade irrestrita requerida pelo usuário, e ainda atenda aos requisitos de segurança da computação pervasiva. Este trabalho se desenvolve dentro do projeto pBuy – portal de compras pervasivo, financiado pela FINEP no período de 2005-2006, e tornou-se necessário uma vez que os sistemas de autenticação existentes não abordam a questão de mobilidade do usuário durante a execução da aplicação. Dessa forma, o presente trabalho é inovador por propor um novo, e eficiente, método de Autenticação para Espaços Pervasivos utilizando credenciais para tal.

Capítulo 1

Introdução

Autenticação é o processo de provar a identidade digital de um usuário ou objeto para uso da rede, aplicação ou recurso. Uma vez autenticado, o usuário pode acessar os recursos baseados em suas permissões estabelecidas através do processo de autorização. Existe uma faixa de técnicas de autenticação desde a simples identificação do usuário baseada em logon e senha, passando por informações biométricas (que distinguem uma pessoa da outra), até sofisticados mecanismos usando *tokens*, certificados digitais e smart cards (algo que somente a pessoa tem). Alguns ambientes de segurança podem requerer um processo de autenticação multifator, combinando, por exemplo, algo que a pessoa sabe (como sua senha) com algo que a distingue (como impressão digital) e algo que ela tem (como um *smart card*).

No cenário delineado pela computação ubíqua ou pervasiva [SAH 2003], chamado espaço pervasivo, há uma diversidade de níveis de segurança que dependem dos requisitos das aplicações que nele executam. Em um espaço pervasivo (também chamado *smart space*), computadores e outros (vários e variados) dispositivos digitais estão totalmente integrados ao ambiente físico do usuário e objetivam auxiliá-lo em suas tarefas diárias. Esse é um ambiente altamente dinâmico e heterogêneo. Nessa visão, o poder computacional está sempre disponível (*always-on*), se encontra em qualquer lugar, acessível a todo momento e com qualquer dispositivo [AUG 2004].

As aplicações pervasivas requerem um novo modelo de projeto e execução para expressar a semântica ‘siga-me’: as aplicações seguem o usuário conforme este se desloca no espaço [AUG 2006]. Desta forma, os recursos, incluindo serviços, dispositivos e aplicações, disponíveis ao usuário podem alterar-se rapidamente. Diferentes espaços têm diferentes tipos de recursos disponíveis e diferentes políticas de uso dos recursos [AUG 2005].

Como a computação pervasiva propõe o deslocamento da visão centrada nos processos computacionais para a visão centrada no usuário e suas atividades, quem deve ser reconhecido pelo sistema é o usuário e não os equipamentos que ele porta ou usa (como estão definidos os sistemas computacionais atuais). Assim, introduzem-se novos requisitos ao projeto de um sistema de

segurança.

O grupo de pesquisa em Sistemas de Computação Móvel (gMob) está, atualmente, desenvolvendo o projeto pBuy – portal de compras pervasivo, com financiamento da FINEP 2005-2006. Nesse escopo, foi projetada a arquitetura de software baseada em serviços que estende o software Portal de Compras da empresa SIG – Soluções em Informática e Gestão, com aspectos de pervasividade.

Um dos serviços necessários é alterar o atual sistema de segurança no acesso ao Portal para atender aos novos requisitos de pervasividade das aplicações do projeto pBuy: independência de dispositivo, mobilidade irrestrita do usuário e centralização nas atividades do usuário. Este trabalho irá, então, projetar e desenvolver essa solução, fazendo as alterações necessárias nas aplicações e serviços já existentes.

Existem inúmeros métodos de autenticação, porém, segundo análise preliminar realizada, nenhum deles preenche de forma satisfatória os requisitos de segurança de sistemas pervasivos. Métodos atuais não suportam a requerida mobilidade do usuário, já que o objetivo é que o usuário possa trocar de dispositivo, devido a uma opção própria ou por possíveis problemas como falta de energia, durante o acesso as aplicações do pBuy sem necessitar refazer todo o processo de autenticação.

Salienta-se que, atualmente, se encontram muitos problemas para fornecer segurança em sistemas de computação pervasiva. Um dos problemas é originário do fato que dispositivos portáteis e embutidos apresentam limitações de recursos, de energia, memória, software e acesso à rede, dificultando o fornecimento de segurança e o gerenciamento desses dispositivos. Outro problema é devido à alteração de foco dos atuais sistemas de autenticação que se baseiam na autenticação de equipamentos para a autenticação centrada no usuário. Como dito, o usuário é quem deve ser autenticado e não o dispositivo que ele está usando no momento, pois ele pode utilizar vários dispositivos durante a execução de uma mesma aplicação, requerendo um processo de autenticação automática. E ainda, os problemas são potencializados devido a grande heterogeneidade de dispositivos móveis, e portáteis, que podem ser utilizados pelos usuários durante o acesso as aplicações. Essas questões potencializam os já complexos requisitos de segurança.

Atualmente, a autenticação dos usuários no Portal de Compras é realizada por logon e senha. Como usuários pervasivos podem usar uma faixa de dispositivos durante o ciclo de vida da aplicação, o mecanismo de autenticação deve ser alterado para atender aos requisitos impostos pela

computação pervasiva, permitindo o acesso de qualquer lugar, a qualquer tempo e com qualquer dispositivo. Além disso, deve-se levar em conta que o usuário não deseja esquemas complicados para acesso ao sistema, uma vez que a entrada de informações via dispositivos móveis não é, ainda, uma interface muito amigável.

Considerando os argumentos acima, faz-se necessário desenvolver uma solução própria, coesa com os serviços já projetados na arquitetura de software pBuy. Com isso propõe-se a implementação de um novo método de Autenticação baseado em credencias para a utilização em Espaços Pervasivos, procurando manter a mobilidade irrestrita requerida pelo usuário e tratando suas conseqüências. Esse serviço procura atender aos requisitos da computação pervasiva em adição aos requisitos de segurança já existentes no Portal de Compras, software-base que está sendo estendido para o escopo pervasivo. A solução proposta é usar credenciais, que são informações dos usuários de interesse da Arquitetura pBuy, baseadas no padrão vCard, e armazenadas juntamente com as outras informações pessoais do usuário contidas neste, de forma a permitir a mobilidade requerida pelo usuário móvel. Dessa forma, o Serviço de Autenticação baseado em Credenciais trata-se de uma inovação propondo um novo e eficiente método de Autenticação para ser utilizado em Espaços Pervasivos.

Vale ressaltar que assim como o Sistema utilizado até então no pBuy para a Autenticação através de logon e senha, o novo modelo de Autenticação baseado em Credenciais não utilizará Criptografia para as informações contidas nas mensagens que são enviadas e recebidas na interação com o usuário. Essa decisão foi tomada considerando o fato que a Criptografia devia ser um recurso fornecido pela Rede e não pelas aplicações, e ainda porque seria muito dispendioso para as aplicações que executam nos dispositivos móveis prover tal recurso de segurança considerando as sérias limitações de hardware que esses dispositivos apresentam.

O restante deste texto apresenta a seguinte estrutura: o segundo capítulo contém a revisão de literatura, onde se apresenta a computação pervasiva, descrevem-se os métodos de autenticação para espaços pervasivos propostos atualmente e uma visão geral da Arquitetura pBuy, descrevendo os serviços que a integram; o capítulo 3 apresenta o foco deste trabalho – serviço de autenticação baseado em credenciais - com uma descrição das tecnologias utilizadas, o detalhamento do processo de autenticação usado atualmente na Arquitetura pBuy e a modelagem do novo Serviço de Autenticação; o capítulo 4 apresenta as conclusões e as considerações finais a respeito deste trabalho.

Capítulo 2

Sistemas Móveis e Autenticação

Este capítulo apresenta a Computação Pervasiva e as limitações impostas pelo ambiente móvel atual, resume o Sistema Kerberos, por ser este muito utilizado para Autenticação em Sistemas Distribuídos, aborda os métodos propostos atualmente para autenticação em espaços pervasivos e, por fim, descreve a arquitetura pBuy e os serviços que a compõem.

2.1 Computação Móvel ou Pervasiva

A computação móvel, tratada por pesquisadores como o novo paradigma do século XXI, é uma área recente de pesquisa que visa disponibilizar a computação aonde o usuário deseja, no momento desejado, o que for desejado e da forma requerida pelo mesmo, através da virtualização de informações, serviços e aplicações.

Diferentemente de como é feito atualmente onde o homem tem que ligar, operar e desligar as máquinas, a computação pervasiva objetiva tornar o uso do computador transparente ao usuário, e dessa forma o homem será rodeado de dispositivos computacionais e estaria interagindo com eles mesmos sem perceber. Com a grande variedade de dispositivos de diversos tipos, móveis ou fixos, aplicações e serviços interconectados, este novo ambiente computacional apresenta limitações e algumas serão citadas na seqüência.

2.1.1 Limitações apresentadas por ambientes móveis

Ambientes móveis atualmente apresentam várias limitações justificadas pelo meio em que operam. Um fator muito importante a ser considerado é a velocidade de processamento de dispositivos móveis, pois o consumo de energia será menor quando maior for a latência que pode ser tolerada no processamento.

Outro fator importante é que as Redes Sem Fio, muito utilizadas nesses ambientes móveis, apresentam largura de banda inferior a redes cabeadas. Dessa forma, é de suma importância que

aplicações pervasivas utilizem a banda eficientemente, e ainda executem em diferentes redes que podem possuir capacidades variadas.

Portanto para mantermos a mobilidade do usuário tão fortemente defendida na computação pervasiva, e ainda tratar problemas como as diversificadas formas de acesso à rede por parte do usuário através de seus dispositivos móveis, novos modelos de organização e acesso a aplicações e aos dados devem ser modelados e implementados.

2.2 Sistema Kerberos

O protocolo de Autenticação Kerberos [KER 2007] foi criado em meados dos anos 80 pelo *Massachusetts Institute of Technology* como parte do Projeto *Athena*, recebendo esse nome com alusão a figura da Mitologia Grega *Cérbero.* Este era um cachorro que possuía três cabeças e rabo de serpente, que evitava a entrada de pessoas ou de coisas indesejáveis nos portões de entrada do Inferno de *Hades.*

Trata-se de um poderoso protocolo para autenticação em arquiteturas cliente / servidor que utiliza chaves secretas compartilhadas criptografadas com o *Data Encrypt Standard* (DES). Dessa forma, para prover o alto nível de segurança, o Sistema compõem-se de dois componentes que atuam independentemente, os quais são tratados na seqüência.

2.2.1 Componentes do Kerberos

O Sistema de Autenticação Kerberos foi estruturado de forma a garantir a maior segurança possível, sendo dividido em dois componentes principais. Estes trabalham de uma forma independente, e assim Kerberos torna-se um Serviço de Autenticação distribuído.

- *Authentication Service* (SA). O Serviço de Autenticação é responsável pela Autenticação do usuário. Ao receber um pedido, este envia ao usuário um *ticket* e uma chave de sessão que serão utilizados para se conectar com o sistema através de um canal seguro;
- *Ticket Granting Service* (TGS). Responsável pela concessão dos *tickets* para os serviços que utilizam o Kerberos. O TGS oferece um *ticket* ao cliente que é utilizado para convencer o servidor de sua autenticidade.

2.2.2 Funcionamento do Sistema Kerberos

O Kerberos, para a autenticação de clientes e servidores, cria mensagens que possuem a chave secreta do usuário e o nome do Servidor de Concessão de *Tickets* (TGS). Este Servidor enviará ao usuário, quando este requisitar, o *ticket* cifrado com a sua chave secreta correspondente. Dessa forma, quando o cliente deseja acessar um serviço, este faz uma requisição de um *ticket* ao TGS correspondente ao serviço desejado. A partir disso, caso esteja correto, é enviado o *ticket* para o cliente que o usará para se relacionar com o Servidor de Autenticação do Kerberos. Este *ticket* é comparado com as informações que o cliente possui como nome, chave secreta, etc., e caso coincidam as informações, os usuários podem se comunicar com o Servidor, ou com outros usuários, através de um canal seguro. Assim, a autenticação no Kerberos implica o conhecimento da chave secreta do cliente, a obtenção de um *ticket* e de uma chave de sessão.

A seqüência de passos do processo é:

- (i) é feita a conexão ao Servidor de Autenticação, onde o usuário é autenticado, recebendo um *ticket* e uma chave de sessão. Esses são criptografadas com uma chave secreta, e são relacionadas entre o cliente e o servidor;
- (ii) o usuário faz uma solicitação de *ticket* ao TGS, onde o *ticket* e a chave de sessão são gerados novamente relacionando o cliente e os serviços que serão acessados com essas informações. Esses dados são criptografados e enviados ao cliente;
- (iii) o usuário envia ao servidor o *ticket* recebido no passo anterior, e como está criptografado com a chave secreta do servidor, este descriptografa a mensagem para ter acesso à chave de sessão que irá ser válida por um tempo limitado;
- (iv) é feito o acesso à base de informações onde são cadastradas e mantidas as chaves secretas do cliente e do servidor;

Dessa forma, o Sistema Kerberos torna-se muito confiável, apresentando um alto nível de segurança, mas tornando-se um sistema complexo. Em virtude disso, e também por não contemplar a visão de mobilidade desejada, onde o usuário pode utilizar vários dispositivos no acesso às aplicações da Arquitetura pBuy sem necessitar refazer o processo de autenticação, Kerberos não foi utilizado como solução de Autenticação. Mas alguns de seus conceitos e estratégias, como a utilização de *ticket* para que o usuário prove a sua identidade ao Sistema de Autenticação, foram utilizadas para o planejamento e estrutura do Serviço de Autenticação baseado em Credenciais.

A seguir, faz-se uma revisão de alguns métodos propostos atualmente para a autenticação em espaços pervasivos.

2.3 Autenticação em espaços pervasivos

Atualmente, muitos métodos são propostos para autenticação em espaços pervasivos, e essas várias abordagens divergem consideravelmente umas das outras em vários aspectos, tais como nível de confiabilidade que proporcionam e robustez. Alguns dos métodos propostos, para o cenário da computação pervasiva, são:

- Métodos tradicionais de autenticação como uso de *logon* e senha, smart cards, informações biométricas, etc. Certificados e assinaturas podem ser úteis para confirmar atributos estáticos como a origem de um objeto e seu tipo;
- Código de Autocertificação. O hardware pode ser configurado para enviar sua configuração, e políticas podem ser implementadas onde dispositivos interagem com outros dispositivos autocertificáveis;
- Agente Móvel atuando em nome do usuário. Permite um certo grau de invisibilidade do processo de autenticação para o usuário final;
- Uso de telefone celular como dispositivos de autenticação [MIN 2003]. O sistema requer que o usuário digite uma senha enviada de seu telefone celular, via mensagem SMS, para o navegador/servidor de autenticação a fim de se logar. Este método considera a autenticação baseada no que a pessoa tem (telefone celular);
- Certificação de atributos associados aos objetos [ZAD 2003]. Os atributos confirmam qual dispositivo é sujeito de uma interação e o que os dispositivos estão fazendo. Existe uma coleção de atributos, tais como localização, tipo de dispositivo, estado do dispositivo, que podem ser autenticados. O contexto e o nível de garantia requerido podem variar dinamicamente dependendo da aplicação e dados envolvidos;
- Uso do conceito de confiança entre usuários para a autorização: se A confia em B, então, B pode acessar os recursos de A enquanto a confiança existir [SAL 2004]. O Projeto SHAD considera que um dispositivo tem sempre um dono (humano ou virtual) e é este que é autenticado. Uma vez que A confia em B, o *Personal Command Module* representa o

usuário no sistema e faz as autenticações automaticamente, negociando um *ticket* para uso dos recursos. A vantagem é que reduz o nível de gerenciamento, resultado importante para ambientes pervasivos e altamente dinâmicos;

- O conceito de federação, um tipo especial de relacionamento de confiança entre limites de redes internas entre organizações distintas, pode também ser útil no ambiente pervasivo [SAL 2004].

Sendo que esses métodos acabam por autenticar os equipamentos utilizados pelo usuário, ou então por não permitir a noção de mobilidade do usuário objetivada para o pBuy, não foram utilizados como solução de autenticação.

Como o trabalho implementa o serviço de autenticação da arquitetura pBuy, a seguir esta é descrita resumidamente.

2.4 Arquitetura de Serviços pBuy

O projeto pBuy, financiado pela FINEP, com período de execução de fevereiro/2005 a março/2007, está sendo desenvolvido pelo grupo de pesquisa Gmob (Grupo de Pesquisa em Sistemas de Computação Móvel – UFSM). O desafio é inserir tecnologia móvel com acesso pervasivo em um sistema legado, chamado Portal de Compras, desenvolvido pela empresa SIG Soluções em Informática e Gestão (www.sigbrasil.com.br). Grande parte das aplicações existentes na área da computação pervasiva é experimental e interna aos grupos de pesquisa. O projeto pBuy é uma oportunidade de avaliar o impacto que essa nova tendência tecnológica traz ao mercado de produção de software.

O software Portal de Compras tem como base um sistema de leilão virtual para a realização de compras. O leilão é dividido em fases, delimitadas por datas ou ações da parte que realiza o leilão. Cada fase gera tanto mensagens do usuário para o sistema quanto mensagens do sistema para o usuário sendo que a última deve ser entregue onde o usuário estiver e no dispositivo em uso no momento por ele [PIR 2005].

Essas características geram a necessidade de um serviço de envio de mensagens (Serviço de Disseminação) para o usuário autenticado pelo sistema (**Serviço de Autenticação**) de forma uniforme e independente do dispositivo que será utilizado. O sistema deve estar ciente do

dispositivo do usuário no momento do envio para fazer a adaptação do conteúdo (Serviço de Apresentação) de forma que o dispositivo aceite a mensagem e disponibilize-a para o usuário, levando em consideração o estado da rede e o tipo de conexão através do Serviço de Comunicação. A figura 2.1 mostra a integração dos serviços para o sistema proposto [SIL 2007], ilustrando também o ciclo normal de funcionamento, e ainda a divisão entre os serviços do lado servidor e cliente.

Primeiramente, quando o serviço de comunicação é ativado no cliente, este mesmo descobrirá, utilizando o protocolo Bonjour [APP 2005], os serviços que compõem a arquitetura pBuy que são os serviços de autenticação, disseminação e o servidor do portal de compras.

O protocolo Bonjour permite criar uma rede instantânea de computadores e outros dispositivos sem a necessidade de predefinir endereços IP ou realizar configurações em servidores DNS. O Bonjour permite que os serviços e as capacidades de cada dispositivo sejam registrados na rede, e que esses serviços sejam descobertos dinamicamente pelos outros dispositivos da rede [RED 2005]. O protocolo Bonjour utiliza as tecnologias de conexão populares, como Ethernet e IEEE 802.11 e sobre o padronizado e amplamente utilizado protocolo IP.

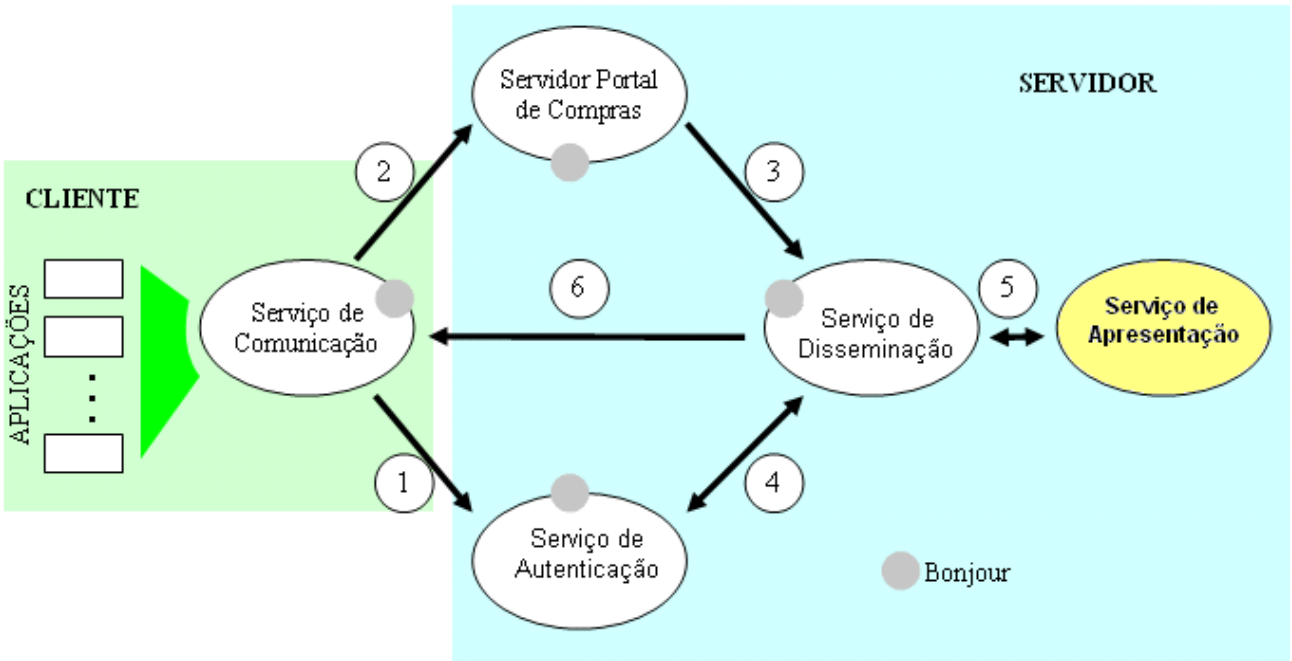

Figura 2.1 Arquitetura pBuy

Após a ativação da comunicação, o usuário deverá, então, autenticar-se no sistema e, com isso, receber o *ticket* que confirma o sucesso na autenticação (1). Em seguida, é solicitada ao

Servidor pBuy, a página inicial do sistema (2). O servidor do pBuy, por sua vez, após consultar o Serviço de Autenticação para obter as informações do usuário que correspondem àquele ticket, envia o documento solicitado (em formato XML – *eXtensible Markup Language*), juntamente com a identificação do usuário destino ao Serviço de Disseminação (3). O Serviço de Disseminação deverá verificar, consultando o Serviço de Autenticação (4), se o usuário está conectado e autenticado. Em caso positivo, o documento XML é enviado ao Serviço de Apresentação (5), para que seja transformado em algo visualizável pelo dispositivo do usuário. Finalmente, as informações solicitadas pelo dispositivo serão entregues pelo Serviço de Disseminação (6).

A comunicação entre os serviços do lado servidor ocorre via RMI *(Remote Method Invocation),* por serem confiáveis entre si e por permitir maior abstração do que o simples envio de mensagens. Nas seções seguintes, os serviços são apresentados com maiores detalhes.

2.4.1 Servidor do Portal de Compras

O Servidor do Portal de Compras é o provedor das informações sobre os leilões virtuais, a comunicação com o sistema legado. É ele quem envia os documentos XML ao Serviço de Disseminação para que sejam adaptados e enviados ao dispositivo do usuário.

Para que as mensagens sejam entregues aos destinatários corretos, o servidor do portal de compras deverá enviar ao serviço de disseminação, a identificação do usuário (através de um identificador provido pelo próprio usuário) e a mensagem.

2.4.2 Serviço de Apresentação de Conteúdo

A adaptação de conteúdo [BEL 2005] permite que informações sejam mostradas corretamente em diversos ambientes. Como a computação pervasiva tem como característica o acesso de qualquer dispositivo, existe a necessidade deste serviço. No sistema pBuy, o serviço de apresentação é utilizado pelo serviço de disseminação de informações que informará o tipo do dispositivo e os dados que sofrerão o processo de adaptação automática. Os dados adaptados ao dispositivo deverão retornar ao serviço de disseminação para envio na forma de mensagem.

O serviço de apresentação deve considerar a heterogeneidade dos dispositivos portáteis e celulares existentes. Como não foram encontradas soluções com as características citadas acima relativas a sistemas móveis, apenas relativas à WEB, concluiu-se que seria necessário desenvolver uma solução própria para atender aos requisitos funcionais do serviço.

Para o desenvolvimento desta solução é utilizada a tecnologia XSL (*eXtensible Stylesheet Language*). O XSL foi desenvolvido pela *Word Wide Web Consortium* (W3C) pela necessidade de uma linguagem de estilos para documentos XML, permitindo assim a transformação de documentos XML em outros tipos de documentos. Neste caso, é armazenada uma única versão com o conteúdo a ser apresentado, enquanto diferentes formas de apresentação são geradas automaticamente a partir deste conteúdo de acordo com o tipo de dispositivo-alvo. Para isso, o armazenamento do conteúdo é no padrão XML de forma a ser convertido no formato apropriado ao dispositivo-alvo.

2.4.3 Serviço de Disseminação de Mensagens

Os usuários da computação pervasiva são móveis e podem explorar as capacidades de vários tipos de dispositivos. O serviço de disseminação de dados [RED 2005] deve ser transparente para quem o utiliza, somente o usuário destino e a mensagem propriamente dita devem ser suficientes para a sua entrega ao dispositivo correto. Assim, o serviço de disseminação deve identificar onde o usuário destino esta, qual equipamento está utilizando no momento e enviar a mensagem de forma adaptada ao dispositivo [RED 2005].

Na Arquitetura de Serviços pBuy, este serviço interage com os outros serviços de suporte às aplicações com comportamento pervasivo, em especial, com o Serviço de Apresentação, o qual realiza a adaptação do conteúdo ao dispositivo utilizado no momento pelo usuário que receberá a mensagem, e o Serviço de Autenticação, para confirmar que o destinatário esteja conectado e autenticado no sistema.

No momento em que receber uma mensagem do Servidor do pBuy, o Serviço de Disseminação consultará o Serviço de Autenticação para saber se o usuário destino encontra-se acessível e autenticado. Em caso positivo, o Serviço de Disseminação envia a mensagem ao Serviço de Apresentação para adaptar a mensagem ao dispositivo do usuário para que ela possa, ser entregue a ele. Caso o usuário não esteja acessível, a mensagem, não adaptada, é armazenada em uma fila de mensagens para ser enviada posteriormente caso não seja do tipo 'imediata', pois esse tipo de mensagem só deve ser enviada naquele exato momento não sendo possível enviá-la posteriormente.

As mensagens vindas do Servidor do pBuy terão um tempo de vida de acordo com sua relevância. Esse tempo de vida determina o tempo em que as mensagens permanecerão na fila de mensagens do Serviço de Disseminação. Por exemplo, mensagens de confirmações de operações ou de erros de validação de dados não necessitam de um tempo de vida muito longo. O usuário não

estaria interessado nessas informações depois de ter passado muito tempo da realização das operações. Já mensagens, por exemplo, de abertura de editais, têm um tempo de vida maior, pois os usuários que a recebem, provavelmente, estarão interessados em participar da licitação. Este aspecto gera a necessidade do armazenamento persistente de algumas mensagens.

2.4.4 Serviço de Comunicação

O serviço de comunicação está sempre disponível no dispositivo móvel (PDA, celular) e é a ponte de comunicação das aplicações do pBuy executando no dispositivo com o mundo externo. Este é responsável por controlar o estado da rede e identificar o tipo de acesso. Estratégias de *caching* são utilizadas para os momentos de desconexão, e ainda é tratado por este serviço uma forma de fornecer as funcionalidades mencionadas considerando as diferentes capacidades de PDAs e celulares.

No próximo capítulo são descritas as características e os aspectos de modelagem do serviço de autenticação.

Capítulo 3

Serviço de Autenticação baseado em Credenciais para Espaços Pervasivos

O Serviço de Autenticação baseado em Credenciais da arquitetura pBuy é o foco deste trabalho, e a seguir apresenta-se um sumário das tecnologias utilizadas na implementação do serviço, e uma descrição sobre modelagem e implementação do Serviço.

3.1 Tecnologias Utilizadas

Para a implementação da arquitetura do sistema pBuy adota-se a plataforma Java, em suas três versões: J2SE, J2EE e J2ME [SUN 2007]. A seguir, descrevem-se os principais conceitos usados para a comunicação: o padrão vCard utilizado para troca de informações entre dispositivos, e a comunicação via RMI entre os dispositivos.

3.1.1 vCard

Para a autenticação na arquitetura pBuy, as aplicações utilizarão credenciais e demais informações do usuário armazenadas no padrão vCard (*www.imc.org/pdi*), o qual visa automatização de troca de informações pessoais comumente encontradas num cartão de identificação usual. Nesse padrão, os dados são apresentados através de meta-informações (*tags*) pré-definidas, responsáveis por dispor os dados de uma forma organizada, facilitando sua utilização. Sendo um padrão internacional mantido pela *Internet Mail Consortium* (IMC) desde 1996, vCard é compatível com as diversas plataformas existentes, tendo como principal foco de interesse os dispositivos móveis, como telefones celular e *Personal Digital Assistants* (PDAs).

Como o padrão vCard é independente de linguagem de programação, a implementação de uma aplicação deve definir um método `vCard-writer`, responsável por criar um arquivo vCard com os dados desejados, e um método `vCard-reader`, responsável por interpretar as informações contidas no cartão eletrônico. Na seqüência segue um exemplo de informações pessoais organizadas e armazenadas no padrão vCard.

```
BEGIN:VCARD
VERSION:2.1
N:Lorenzi da Silva;Fábio
FN:Fábio Lorenzi da Silva
NICKNAME:lorenzi
ADR;HOME:;;Av. Júlio de  Castilhos;Restinga Sêca;RS;97200-000;Brasil
BDAY:20070301
EMAIL;PREF;INTERNET:lorenzi@inf.ufsm.br
END:VCARD
```

O padrão vCard permite extensão de *tags*, atendendo ao objetivo de permitir uma maior flexibilidade no tratamento de informações dos usuários e adequação às necessidades de cada aplicação. Uma característica importante é que essa extensabilidade não afeta o padrão original convencionado, pois as informações não reconhecidas são ignoradas pelo interpretador, sendo somente tratadas as informações pré-estabelecidas no padrão vCard. Desse modo, a garantia de unicidade se mantém entre diferentes aplicações que envolvam o intercâmbio de um mesmo cartão de identificação eletrônico.

O padrão vCard foi estendido para conter além das informações pessoais do usuário, as credenciais de interesse das aplicações da arquitetura pBuy, as quais refletem uma política de controle de acesso baseada em papéis do usuário. Decisões de acesso serão baseadas nos papéis que os usuários têm como parte do sistema Portal de Compras: administrador, licitante, fornecedor, etc. Usuários podem ter diferentes privilégios de acesso dependendo do nível de confiança necessário para acesso aos recursos e aplicações. Esses papéis são estabelecidos pela administração da aplicação e podem variar no tempo. Correntemente, somente são autenticados usuários conhecidos e cadastrados no Portal de Compras.

Para a utilização do padrão em dispositivos móveis, a API Java Microedition fornece vários métodos para a manipulação de informações pessoais. Isso é fornecido na API `javax.microedition.pim` onde é disponibilizado a classe `Contact` que representa e organiza as informações pessoais no padrão vCard. A partir disso, implementou-se mecanismos que criam, atualizam e removem informações de contatos, e ainda modos para transformar um `Contact` em `OutputStream` e para a partir de um `InputStream` obter o `Contact` necessários para a transmissão de dados via Java Socket . As classes e métodos estão ilustradas na figura 3.1.

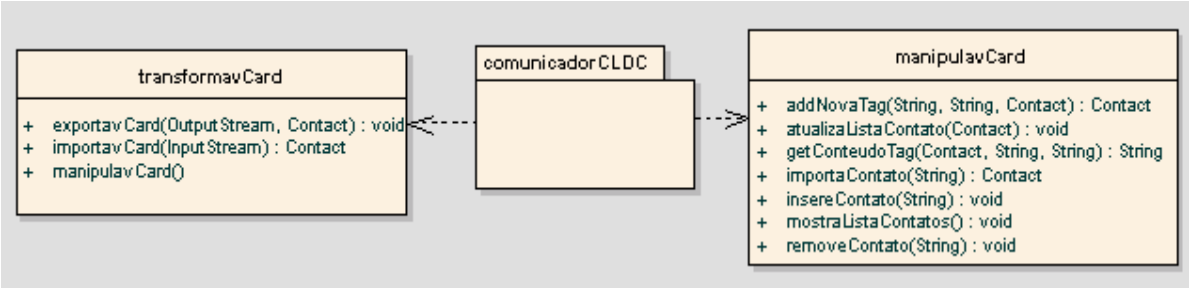

Figura 3.1 Diagrama de Classes do uso do vCard no Comunicador

No módulo de Autenticação, foi necessário implementar métodos que manipulam o padrão vCard e com isso permitir a aplicação compreender as informações contidas nele. Com isso, criaram-se funções responsáveis por extrair informações das *tags* do padrão, e ainda inserir novas *tags* com os valores desejados. A seguir, apresentam-se tais funções.

```
public String getConteudoTag (String vCard,String nomeTag,int tipoTag) {
       String valorTag;
       int inicio= 0; int fim = 0;
       nomeTag = nomeTag.toUpperCase();
       inicio=vCard.indexOf(nomeTag + ":");
       if (inicio == -1) {
            System.out.println("Tag " + nomeTag + " nao existe.");
            return null;
        }
        inicio++;
        valorTag=vCard.substring(inicio);
        inicio = valorTag.indexOf(":") + 1;
        if (tipoTag==1)
            fim=valorTag.indexOf("\n");
        else
            fim = valorTag.indexOf("--");
        if (fim == -1) {
            fim = valorTag.length();
        }
        valorTag = valorTag.substring(inicio, fim);
        return valorTag;
    }
```

O método acima é responsável por obter o conteúdo de *tag's* contidas no padrão vCard. Ele recebe como argumento o vCard, o nome da *tag* que se deseja obter a informação e o tipo dela, para diferenciar as *tag's* pré-definidas do padrão e as credenciais estendidas para o pBuy. O método, após executar suas operações, retorna o conteúdo da *tag* desejada.

Este outro método abaixo estende o padrão vCard inserindo novas credenciais necessárias para o Projeto pBuy. Este recebe como argumento o vCard recebido do usuário, o nome da credencial que vai ser acrescentada, e o conteúdo da mesma. Dessa forma, o método realiza as operações necessárias e retorna a string com o vCard estendido com a nova informação.

```
public String addTag(String vCard, String nomeTag, String conteudoTag) {
        String valor = "";
        String auxInicio,auxFim;
        int inicio;
        nomeTag = nomeTag.toUpperCase();
        valor=" "+ nomeTag+ ":";
        inicio=vCard.indexOf(valor);
        if (vCard.contains(valor)) {
            String valorAntigo=getConteudoTag(vCard,nomeTag,2);
            inicio=vCard.indexOf(valor);
            auxInicio=vCard.substring(0,(inicio));
            auxFim=vCard.substring((valorAntigo.length())+inicio+(nomeTag.length
            ())+4);
        } else {
            inicio=vCard.indexOf("NOTE");
            String note=getConteudoTag(vCard,"NOTE",1);
            auxInicio=vCard.substring(0,((note.length())+inicio+5));
            auxFim=vCard.substring((note.length())+inicio+5);
        }
        return (auxInicio+valor+conteudoTag+" --"+auxFim);
}
```

3.1.2 RMI

A Invocação Remota de Métodos Java (RMI) [SUN 2006] é uma versão orientada a objetos da *Remote Procedure Call* (RPC), e a sua utilização permite que um objeto invoque o método de outro objeto que pode estar localizado em um *host* diferente, de maneira transparente como se fosse feita a chamada a um método local. Para isso, é necessário que os programas Java obtenham referência dos métodos contidos nos objetos remotos, o que é realizado pelo serviço de nomes do RMI Java.

3.2 Módulo de Autenticação

Uma primeira versão do serviço de autenticação da arquitetura pBuy havia sido projetada para realizar a autenticação dos usuários cadastrados no sistema através de logon e senha. Esse sistema ainda armazena informações como o tipo do dispositivo utilizado pelo usuário na seção atual, o estado da conexão. Essas informações são disponibilizadas aos outros serviços da Arquitetura pBuy, como ao Disseminador de Mensagens e o de Apresentação de Conteúdo. A informação sobre o estado da conexão do usuário é usada pelo Serviço de Disseminação a fim de que ele possa verificar a disponibilidade do usuário, para então decidir se deve enviar a mensagem, ou então armazená-la para envio posterior. O tipo de dispositivo auxilia o Serviço de Apresentação para a adaptação dos documentos a um tipo suportado pelo dispositivo.

No processo de autenticação de um usuário, e no caso de sucesso do processo, o Serviço de Autenticação gera um *ticket* que identifica aquela seção e o envia ao Serviço de Comunicação no

dispositivo do usuário. O *ticket*, que é uma `String`, é gerado de forma aleatória pelo Serviço de Autenticação e vai ser válido por um determinado tempo, que é definido pelo Serviço de Autenticação. Após esse tempo, o *ticket* perde a validade e é gerado outro novo para o usuário. Esse *ticket* é utilizado para o envio ao Servidor do Portal de Compras a cada requisição por questões de segurança, pois assim o usuário será identificado pelo sistema como autenticado. Dessa forma, o Servidor do pBuy tem condições de certificar-se que o *ticket* é válido e solicitar informações do usuário e suas permissões consultando o Serviço de Autenticação.

Para manter a informação do estado da conexão do dispositivo, o Serviço de Comunicação, residente no dispositivo, envia periodicamente mensagens para sinalizar que permanece conectado ao sistema, os *heartbeats* [AGU 97]. O não recebimento por parte do serviço de autenticação, determinado por um *timeout,* caracteriza o dispositivo como desconectado. O Serviço de Autenticação responde a cada uma das mensagens enviadas pelo Serviço de Comunicação e, assim, esse pode saber que continua conectado ao sistema. A resposta enviada pelo Serviço de Autenticação pode ser uma confirmação de que o dispositivo está autenticado ou, caso contrário, a não confirmação sinalizando que o Serviço de Comunicação de reenviar os dados de identificação.

Vale ressaltar que o Serviço de Autenticação baseado em Credenciais utiliza as mesmas políticas de *ticket* e de estado de conexão do dispositivo definidas no Serviço de Autenticação através de *logon* e senha.

3.3 Estrutura do Serviço de Autenticação baseado em Credenciais

Uma nova versão do serviço de autenticação expande a primeira versão para permitir uma flexibilidade maior no processo. O pacote de autenticação baseado em Credenciais é composto por várias classes, figura 3.2, que foram implementadas para prover as funcionalidades necessárias, e interagir com as já existentes no processo de autenticação implementado até então. Na seqüência essa estrutura de classes é tratada.

3.3.1 Estrutura para a Autenticação na base de informações

O Sistema de Autenticação baseado em Credenciais utilizou a classe **Autenticador,** que já estava implementada, sendo necessário o acréscimo de novos métodos para permitir também a sua utilização para a Autenticação através de Credenciais. Essa classe contém as informações de

usuários ativos e inativos no sistema, e essas estão organizadas através do tipo `hashMap`. A escolha desse tipo deve-se ao fato de propiciar acesso rápido as informações já que utiliza-se uma chave, definida pelo programador, para acesso as informações. Existem três hashMap's no sistema sendo duas para os usuários ativos, diferenciando-se na chave de acesso, e outra para os inativos. São elas:

- usuariosAtivosPorId: `hashMap` em que a chave de acesso é o Id (*ticket*). Após a autenticação no sistema, o Comunicador envia periodicamente o seu *ticket* para o Serviço de Autenticação que de posse deste pode acessar as informações do usuário de uma forma otimizada, e também facilita o "descobrimento" de a qual usuário pertence aquele *ticket* recebido.
- usuáriosAtivosPorLogin: `hashMap` onde a chave de acesso é o *login* do usuário. Facilita o acesso às informações no processo de autenticar um usuário no sistema tanto por *login* e senha, quanto por Credenciais;
- usuáriosInativos: utilizada para manter os usuários inativos e a sua chave de acesso é o *login*.

Além disso, o **Autenticador** fornece métodos para a autenticação no sistema, e ainda para o gerenciamento de *timeout* relacionado a cada usuário. Para isso, instancia uma classe `TimerTask` que gerencia o "tempo de vida" do usuário, e após o término deste, move o usuário para os Inativos e solicita ao mesmo o envio das credenciais para refazer o processo de autenticação.

3.3.2 Estrutura para o Tratamento das Credenciais

Para o recebimento e extração das credenciais contidas no vCard do usuário, implementou-se várias classes que apresentam funcionalidades específicas. A classe **manipulaMensagens** é responsável por todas as formas de acesso ao vCard, já que é ela que "entende" a forma de organização das informações no padrão. Nesta classe existem métodos para manipular as mensagens recebidas do Comunicador, outros para adicionar novas *tag's*, e também para se obter o conteúdo das mesmas. Todos esses métodos respeitam o padrão vCard.

A classe **tratavCard** é instanciada pela **recebevCard**, responsável pela criação do Servidor Java Socket, para o recebimento das credenciais enviadas pelo usuário. Ainda realiza o tratamento do vCard, estendido com as credenciais de interesse da aplicação, e também as operações que devem ser tomadas de posse dos dados. Para isso, instancia algumas classes que são:

- **Autenticador:** fará as operações devidas para inserir o usuário nas `hashMap` dos usuários ativos, e também criar a classe que implementa o `TimerTask` para o tratamento do *timerout*;

- **manipulaMensagens:** responsável por manipular as credenciais recebidas e enviadas na interação com o Serviço de Comunicação, e nas trocadas com o banco de dados;

- **trataInformacoesBD:** utilizado para o tratamento das informações que são trocadas com o repositório de credenciais;

O repositório de Credenciais utilizado pelo Serviço de Autenticação baseado em Credenciais é tratado na próxima seção.

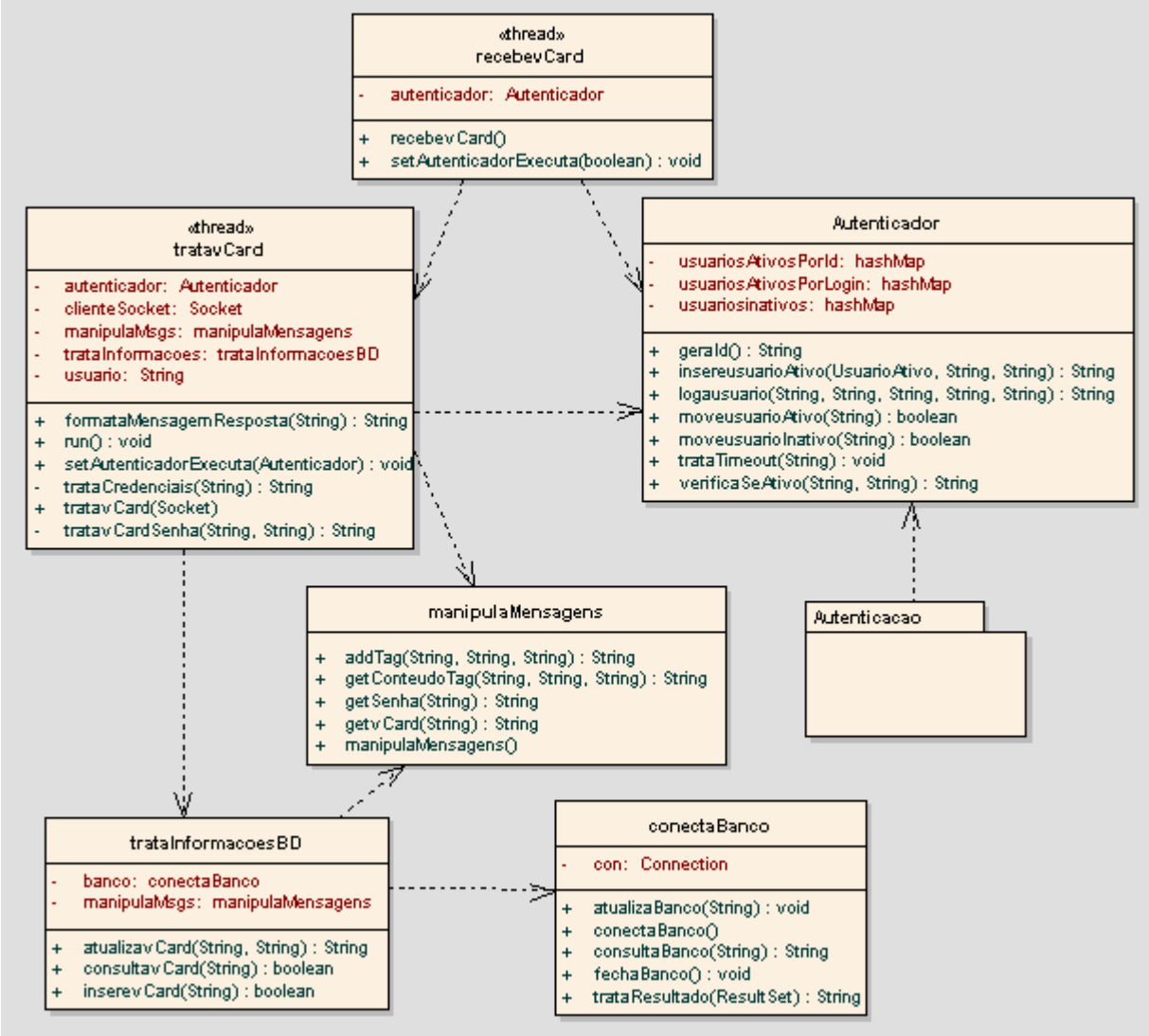

Figura 3.2 Diagrama de Classes do Serviço de Autenticação através de Credenciais

3.3.3 Repositório de Credenciais

Considerando que alguns objetivos do Serviço de Autenticação baseado em Credenciais são autenticar o usuário e não os equipamentos que ele utiliza no momento da autenticação e assim permitir que este possa trocar de dispositivo durante o acesso ao pBuy sem que necessite refazer o processo de autenticação novamente, criou-se o repositório de vCard's.

Na seqüência são tratas as seqüências de ações de todo o processo que engloba a Autenticação na Arquitetura pBuy através de Credenciais.

3.4 Seqüência de ações do processo de Autenticação

Nessa seção descreve-se todos os passos para o processo de Autenticação através de Credenciais no qual interagem o Serviço de Comunicação e o Serviço de Autenticação.

Dessa forma, o processo inicia-se no Serviço de Comunicação e é descrito na seqüência.

3.4.1 Seleção e envio das Credenciais

O processo de Autenticação onde o usuário prova a sua identidade digital para o Sistema, e no caso utiliza Credenciais, inicia no Serviço de Comunicação onde são feitas a seleção e envio das Credenciais. Assim, a seqüência para o processo, figura 3.3, é:

- (i) o usuário que está utilizando o Serviço de Comunicação, informa o *login* ou alguma outra palavra chave que será utilizada para a busca das Credenciais nos Contatos armazenados no sistema;
- (ii) caso o sistema encontre o Contato estendido com as credenciais, é estabelecida a comunicação Socket com o Módulo de Autenticação, e após o contato é transformado para o objeto `OutputStream` necessário para a comunicação.
- (iii) posteriormente é feito o envio do vCard, através do `OutputStream`, para o módulo de Autenticação;
- (iv) caso o a busca pelo vCard não retorne nenhum Contato, é disponibilizado ao usuário uma interface para a inserção das informações e a partir disso é criado um contato;
- (v) é estabelecido a comunicação Socket com o Módulo de Autenticação, e transforma-se o

contato no objeto necessário para o envio;

- (vi) o comunicador envia ao Serviço de Autenticação baseado em Credenciais o vCard;

Após o comunicador enviar as Credenciais, o Serviço de Autenticação recebe a mensagem do usuário e inicia as suas operações que são descritas na seqüência.

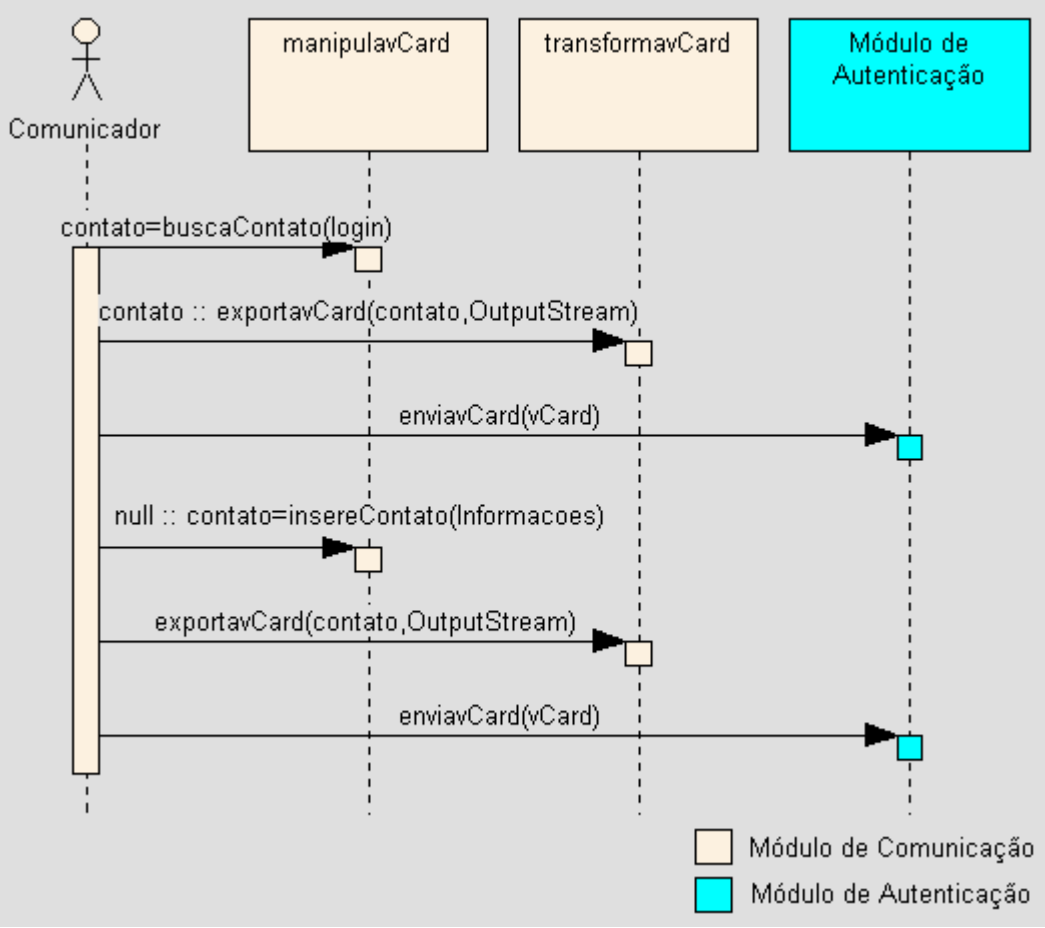

Figura 3.3 Seleção e envio do vCard

3.4.2 Recebimento das Credenciais

O Serviço de Autenticação baseado em Credenciais, executa a classe, que é uma Thread, **recebevCard** que inicializa um Servidor Socket, e após estabelecida a conexão com o cliente aguarda o envio de alguma mensagem.

Na ocasião do recebimento da mensagem, figura 3.4, executa os seguintes passos:

- (i) o **recebevCard** seleciona as credenciais de interesse da aplicação, contidas no vCard enviado pelo usuário, e verifica se o usuário está ativo;

- (ii) o **Autenticador** verifica na sua base de informações e estando o usuário ativo, e de acordo com a necessidade atualiza o tipo de dispositivo do usuário utilizando o servidor DNS para obter esta informação. Após é liberado o acesso ao sistema baseado nas credenciais exigidas pela aplicação;
- (iii) se o usuário não está ativo, o processo de autenticação reinicia: a senha é solicitada;
- (iv) o usuário informa a senha ao **recebevCard** e este tenta se autenticar;
- (v) se os dados conferirem, o usuário é ativado pelo **Autenticador** e tem acesso ao sistema;
- (vi) senão, a senha é solicitada novamente, repetindo o processo;
- (vii) novamente repete-se o processo, e no caso das informações necessárias forem confirmadas, o **Autenticador** ativa o usuário.

No caso de o sistema autenticar o usuário, é gerado um *ticket* de validade do processo de autenticação, conforme as necessidades de segurança da aplicação. Enquanto o *ticket* não expirar, o módulo de autenticação autoriza o acesso do usuário às aplicações automaticamente, sendo que o *ticket* é uma credencial temporal, apresentando um tempo de validade, armazenada juntamente com as outras credenciais de interesse da aplicação contidas no vCard do usuário. Para isso, o módulo de autenticação deve interagir com o repositório de Credenciais, processo descrito na seção subseqüente, para então enviar o vCard estendido com as credenciais ao Comunicador.

O vCard do usuário pode conter informações adicionais necessárias para autenticação em outros sistemas, permitindo assim que o usuário não tenha que memorizar para cada sistema as suas credenciais, somente a sua senha que pode ser interativamente relembrada através de dicas cadastradas previamente no vCard.

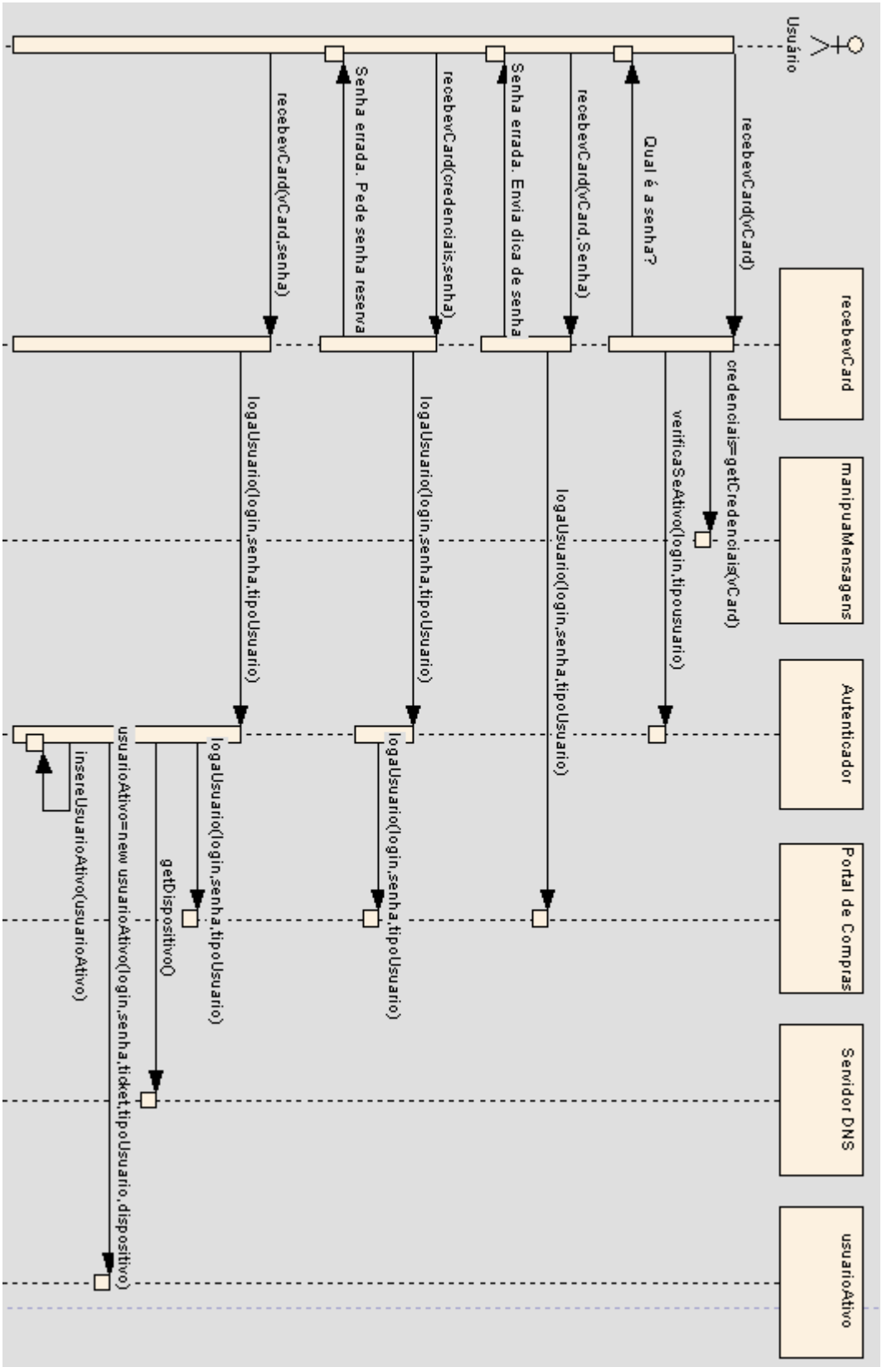

Figura 3.4 Primeiras ações no módulo de Autenticação

3.4.3 Atualização das Credenciais no Sistema

Considerando o fato que o sistema de Autenticação deve tratar possíveis trocas de dispositivos feita pelo usuário durante o acesso ao sistema, utiliza-se o repositório de Credenciais e o sistema interage com ele devido aos seguintes fatores:

- O vCard estendido com as credenciais que o usuário possui armazenadas no seu dispositivo, utilizado no momento, podem não ser as mais atualizadas. Isso pode ocorrer no caso do usuário estar autenticado e precisar trocar de dispositivo, este pode não conter as credenciais atuais do usuário e então o Sistema de Autenticação deve tratar essa situação e permitir o acesso às aplicações. Isso deve ocorrer considerando o *timeout* associado ao usuário, que indicará quando este deve enviar as credencias novamente comprovando a sua identidade. Nesse caso o **Autenticador** deve ainda atualizar as credenciais com o novo dispositivo utilizado e inserir essas informações no repositório;
- no caso do usuário estar se autenticando no sistema, o **Autenticador** gera um *ticket* para este, e ainda deve atualizar essa e outras possíveis informações, como o dispositivo utilizado no processo, no repositório;

Assim a seqüência do processo, figura 3.5, é:

- (i) após o usuário ser ativado no Sistema, o **Autenticador** realiza uma consulta no repositório para verificar se o usuário possui credenciais armazenadas na base de dados;
- (ii) caso exista, então é feito a atualização das credenciais armazenadas no repositório, e posteriormente envia-se ao usuário o vCard estendido com as credenciais atualizadas;
- (iii) se o usuário ainda não possuir credenciais no repositório, então insere no banco as mesmas e envia-se o vCard estendido com as credenciais atualizadas ao usuário;

Após o envio do vCard ao Serviço de Comunicação, este executa várias operações que serão descritas na seqüência.

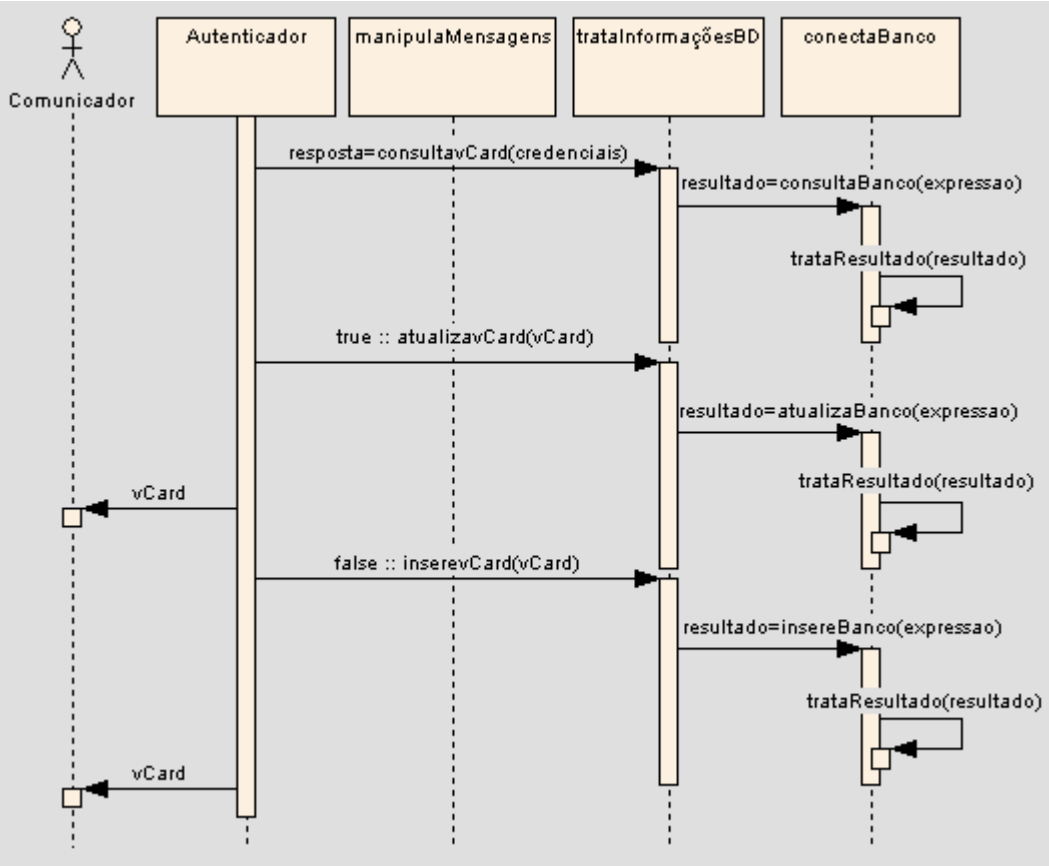

Figura 3.5 Interação com o repositório de Credenciais

3.4.4 Tratamento das credenciais no Comunicador

Como etapa final do processo, o Serviço de Comunicação recebe do Serviço de Autenticação baseado em Credencias o vCard estendido com as credenciais de interesse do pBuy, sendo o *ticket* também considerado uma credencial. Dessa forma, a seqüência, figura 3.6, é:

- (i) Após a mensagem ter sido recebida pelo Comunicador, esta é transformada no formato Contato que é disponibilizado pela API `javax.microedition.pim`;
- (ii) O contato com as credenciais atualizadas e enviadas pelo Serviço de Autenticação é armazenado, e também é retirado o *ticket* das credenciais;
- (iii) De posse do *ticket* o comunicador cria o **Daemon** que é responsável por ficar enviando periodicamente o *ticket* para o Serviço de Autenticação;

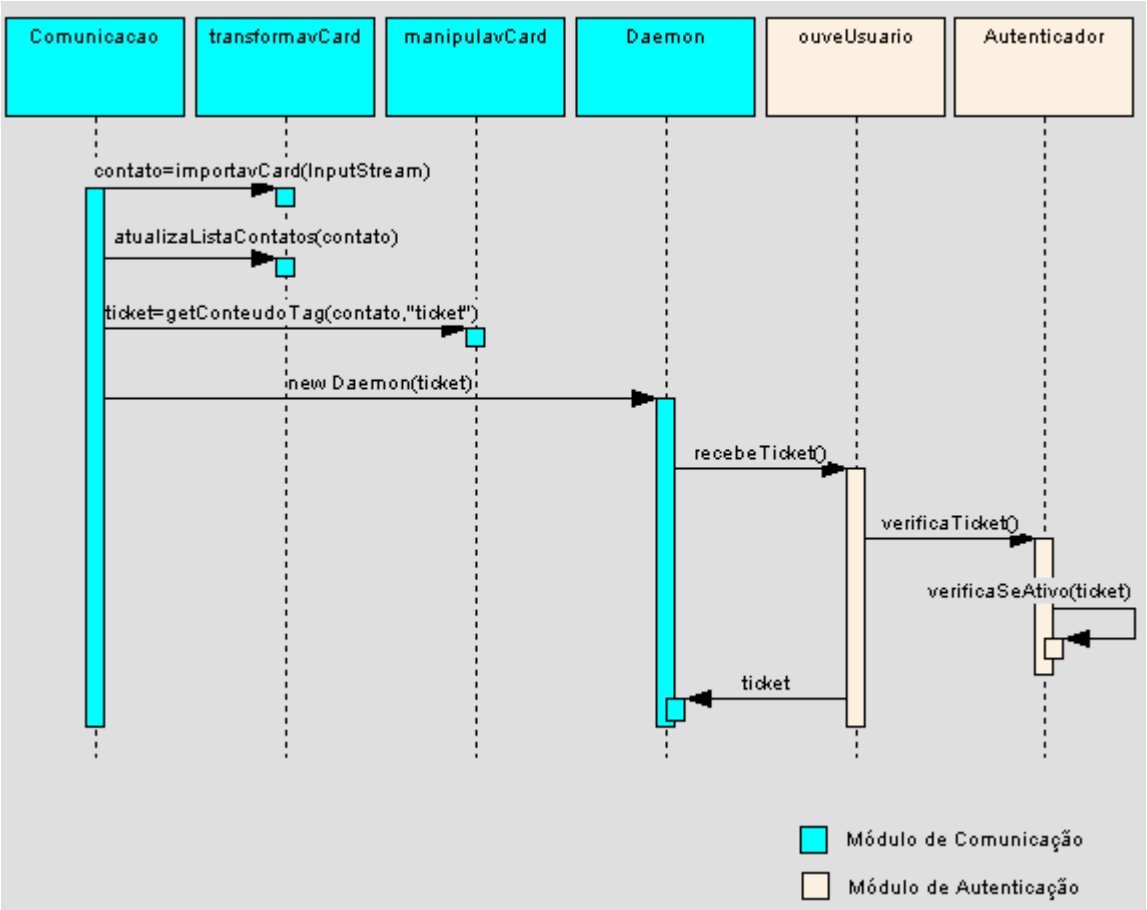

Figura 3.6 Interação Comunicação / Autenticação através do *ticket*

- (iv) a classe **ouveUsuario** responsável pelo recebimento do *ticket* enviado pelo usuário, recebe a mensagem e verifica se o usuário a qual pertence o *ticket* está na base de informações de usuários ativos (se o *ticket* é válido). Essa classe ainda envia ao **Daemon** no comunicador o *ticket* no formato "ok;*ticket*" caso o ticket seja válido ainda, ou então "nok;*ticket*" caso o *ticket* tenha expirado (*timeout estourou*, e o usuário passa a ser inativo). Nesse caso, o processo de autenticação deve ser reiniciado para então o usuário receber um *ticket* válido;

As etapas (iii) e (iv) são executadas periodicamente até o momento que o Serviço de Comunicação se deslogar do sistema e parar sua execução.

3.5 Testes e resultados

Foram realizados vários testes para verificar as funcionalidades do Serviço implementado. Os testes foram tanto locais, testando o módulo de autenticação em si, quanto os que integravam o Serviço de Autenticação baseado em Credenciais com os outros Serviços da Arquitetura pBuy. Os testes de integração apresentaram três etapas diferentes que são:

- (i) integração com o processo de autenticação já existente que se baseava em *logon* e *senha*. Dessa forma, pôde-se testar e concluir que os dois métodos de autenticação puderam ser integrados, e ainda podem utilizar à mesma base de informações que armazena os usuários ativos e inativos no sistema;
- (ii) integração do novo modo de Autenticação com o Serviço de Comunicação e fazer este se autenticar utilizando as Credenciais estendidas no vCard. O Serviço de Comunicação autenticou-se normalmente no novo método e pôde acessar as aplicações;
- (iii) integrar com os outros módulos da Arquitetura, como o Disseminador, que utilizam informações do Autenticador para verificar se o usuário está ou não autenticado. Dessa forma, comprovou-se que o Serviço de Autenticação baseado em Credenciais provê corretamente as informações ao outros módulos da Arquitetura.

Após as etapas de Integração do Sistema, foram realizados os testes das funcionalidades esperadas para o novo Serviço de Autenticação. Como um dos principais objetivos do sistema era permitir a mobilidade do usuário, podendo este trocar de dispositivo durante o acesso às aplicações, testou-se essa funcionalidade. O Serviço de Autenticação baseado em Credenciais comportou-se da forma esperada, sendo que após um usuário estar autenticado, pôde acessar as aplicações da arquitetura pBuy com o envio do vCard estendido com as Credenciais e não necessitou refazer todo o processo de Autenticação.

Além disso, como o sistema de Autenticação gera *tickets* aleatórios para os usuários tentou-se gerar possíveis *tickets* válidos e correspondentes a usuários ativos naquele instante, e ainda reutilizar *tickets* que já tinham pertencido a usuários e que tinham expirado, para tentar burlar o sistema e obter a liberação para acessar as aplicações. Os testes foram realizados e não se conseguiu sucesso na tentativa de fraude, mas mesmo assim não se pode garantir que ataques como esses nunca obterão sucesso (tanto que melhorias na geração e no ciclo de vida do *ticket* são propostas na seqüência do trabalho como trabalhos futuros). Isso se deve ao fato que mesmo sendo pequena, existe a possibilidade de obter-se um *ticket* que corresponda a algum dos usuários ativos no instante.

Vale ressaltar que os testes realizados com o Serviço de Comunicação, para a Autenticação através de Credenciais, foram feitos com o simulador para J2ME presente no *Wireless ToolKit*. Dessa forma, como teste, futuramente pode ser utilizado um aparelho real de Celular para analisar o comportamento dos Serviços envolvidos na autenticação, e até mesmo o desempenho num dispositivo real do processo de Autenticação.

Os testes apresentaram resultados satisfatórios, comprovando o correto funcionamento de aspectos importantes do Serviço de Autenticação baseado em Credenciais.

Capítulo 4

Conclusão

Ambientes pervasivos do futuro próximo envolvem interação, coordenação, cooperação de numerosos, casualmente acessíveis e, frequentemente, invisíveis dispositivos e serviços. Esses dispositivos podem ser carregados por pessoas ou localizados/embutidos no espaço pervasivo e conectados por redes cabeadas ou sem fio, ligando-os uns aos outros e fornecendo informações e serviços integrados. No entanto, apesar dos avanços realizados no hardware e software no sentido de tornar realidade a computação pervasiva, muito ainda falta para que esta se torne presente no nosso dia-a-dia.

Previsto para ser a computação do século 21, somente agora este cenário computacional começa a ser explorado. Resultados iniciais de projetos de pesquisa identificaram os requisitos e questões a serem resolvidas para a concretização desse ambiente computacional.

Os problemas de segurança existentes em sistemas distribuídos tradicionais ficam potencializados neste novo cenário. Além disso, há a adição da necessidade de autenticar e autorizar novas entidades/objetos e deslocar o processo de segurança para centralizar-se no usuário e não nos dispositivos, como ocorre hoje.

Este trabalho apresenta a implementação e utilização de um Serviço de Autenticação baseado em Credenciais para Espaços Pervasivos, sendo que é uma solução própria, coesa com os serviços já projetados na arquitetura de software pBuy. Dessa forma, implementou-se um novo método de Autenticação que não restringe a mobilidade requerida pelo usuário e ainda trata suas conseqüências. Esse Serviço atendeu aos requisitos da computação pervasiva e ainda aos requisitos de segurança já existentes no Portal de Compras.

O Serviço de Autenticação baseado em Credenciais para Espaços Pervasivos foi desenvolvido utilizando-se a tecnologia *Java 2 Standard Edition* (J2SE) juntamente com o padrão vCard (padrão de gerenciamento de informações pessoais).

O trabalho realizado deixa margem para melhoramentos e trabalhos futuros. Entre eles, citam-se:

- Estudar a possibilidade e viabilidade de aperfeiçoar estratégias de segurança do Serviço de Autenticação para a plena utilização em Espaços Pervasivos, já que se trata de uma solução própria que contempla os requisitos de segurança da Arquitetura pBuy;

- Estudar a possibilidade e viabilidade de utilização de mecanismos de Criptografia para as trocas de mensagens do Serviço de Comunicação e o Serviço de Autenticação baseado em Credenciais, devido às sérias limitações de recursos apresentadas por dispositivos móveis. Dessa forma, pode-se diminuir possíveis problemas de interceptações de informações por usuários maliciosos;

- Melhorar as políticas de geração, e do ciclo de vida, dos *tickets*, e assim garantir mais fortemente que usuários indesejados não venham a ter sucesso na tentativa gerarem *ticket*'s válidos e conseguirem acessar o sistema;

- Expandir a utilização de Credenciais para que essas sirvam também para criar e manter Perfis dos Usuários. Até então as credenciais são utilizadas para a Autenticação no Sistema, mas pode-se também associa-las, juntamente com as outras informações do vCard do usuário, a perfis dos usuários que utilizam o pBuy. Dessa forma, tendo um perfil definido para os usuários, várias estratégias de segurança podem ser criadas e utilizadas baseando-se nessas informações, e ainda serem criadas outras aplicações da Arquitetura pBuy que podem usufruir dessas informações.

O trabalho foi concluído atingindo satisfatoriamente os objetivos propostos. Suas contribuições ocorrem tanto na conclusão do projeto, mas também como fonte bibliográfica no desenvolvimento de trabalhos futuros que contemplem, mesmo que tangencialmente, alguns dos aspectos abordados.

Referências

[AGU 97] AGUILERA, M. K.; CHEN, W.; TOUEG, S. Heartbeat: A timeout-free failure detector for quiescent reliable communication. In: LECTURE NOTES IN COMPUTER SCIENCE: DISTRIBUTED ALGORITHMS, PROC. OF 11TH INTERNATIONAL WORKSHOP, WDAG'97, 1997, Saarbrucken, Germany. **Anais...** Springer, 1997. v.1320, p.126-140.

[AUG 2006] AUGUSTIN, I. **ISAMadapt: Abstractions and Tools for Designing General-Pupose Pervasive Applications, Software: Practice and Experience – Special Issue on Auto-adaptive and Reconfigurable Systems. 2006.** Wiley & Sons Publisher (to appear).

[AUG 2005] AUGUSTIN, I. **Managing the Follow-me Semantics to Build Large-scale Pervasive Applications. 2005.** Middleware Conference 2005. Workshop on Middleware for Ad-hoc and Pervasive Computing. Grenoble, France.

[AUG 2004] AUGUSTIN, I. **ISAM Joing Context-awareness and Mobility to Building Pervasive Applications. 2004.** In: Mobile Computing Handbook. Mahgoub, I. and Ilyas, M. Editors, CRC Press, New York.

[APP 2005] APPLE developer connection – bonjour. http://developer.apple.com/networking/bonjour/index.html – acessado em agosto/2006

[BEL 2005] BELUSSO, R. C. **Serviço de Apresentação Consciente do Dispositivo em um Ambiente Pervasivo. 2005**, Monografia (Graduação em Ciência da Computação) – Universidade Federal de Santa Maria.

[HSQ 2006] Hsql Database Engine. **Hsqldb.** http://hsqldb.org/ - acessado em setembro / 2006.

[KER 2007] MIT Kerberos. **Kerberos:The Network Authentication Protocol.** http://web.mit.edu/Kerberos **-** acessado em Fevereiro / 2007.

[MIC 2003] MICHALAKIS, N. **Location-aware Access control for Pervasive Computing**

Environments. 2003. Master Thesis, Massachusetts Institute of Technology. Disponível em www.org.lcs.mit.edu/pubs/michalakis.pdf.

[MIN 2003] MIN WU. **Secure web Authentication with Mobile Phones. 2003.** MIT Oxigen Workshop. Disponível em www.mit.edu/~minwu.

[PIR 2005] PIRES, R. **Serviço de comunicação consciente do estado da rede em um ambiente pervasivo. 2005.** Monografia (Graduação em Ciência da Computação) – Universidade Federal de Santa Maria.

[RED 2005] REDIN, R. M. **Serviço de suporte a disseminação de informações independente de dispositivo em um ambiente de computação pervasiva. 2005**. Monografia (Graduação em Ciência da Computação) – Universidade Federal de Santa Maria.

[SAL 2005] SALBADOR, E. S. **SHAD: A Human Centered Security Architecture for Partitionable, Dynamic and Heterogeneous Distributed Systems. 2005.** 1st ACM International Middleware Doctoral Symposium, Toronto, Canada.

[SAH 2003] SAHA, D. and MUKHERJEE, A. Pervasive Computing: a Paradigm for the 21st Century, IEEE Computer, New York: **IEEE Computer Society**, v.36, n 3, p.25 – 31, mar.

[SUN 2007] SUN Microsystems. **Java 2 Software The Platforms**. Disponível em http://java.sun.com/java2/index.html, acessado em março de 2007.

[SUN 2006] SUN microsystems. Java Remote Method Invocation (Java RMI). http://java.sun.com/products/jdk/rmi, acessado em outubro/2006.

[ZAL 2003] ZALIUDDIN, I. **Authentication in Pervasive Computing: Position Paper. 2003**, 1st International Conference on Security in Pervasive Computing, Germany, mar.

Apêndice A. Dicionário das classes do Serviço de Autenticação baseado em Credenciais

autenticacao

Class Autenticador

```
java.lang.Object
  java.rmi.server.RemoteObject
    java.rmi.server.RemoteServer
      java.rmi.server.UnicastRemoteObject
        autenticacao.Autenticador
```

All Implemented Interfaces:

java.io.Serializable, java.rmi.Remote

```
public class Autenticador
extends java.rmi.server.UnicastRemoteObject
```

Classe responsável por armazenar e pelas operações de manutenção da base de informações de Usuários Ativos e Inativos no sistema.

See Also:

Serialized Form

Field Summary

Fields inherited from class java.rmi.server.RemoteObject

`ref`

Constructor Summary

`Autenticador()`

Método construtor da classe Autenticador

Method Summary

`boolean`	`autenticado(boolean t)` Responsável por verificar se esta ativo
`boolean`	`estaAtivo(java.lang.String login, java.lang.String tipoUsuario)` Método usado pelos outros Serviços da arquitetura para saber se o usuário esta ativo ou não
`java.lang.String`	`geraId()` Responsável por gerar o ticket que será o identificador do usuário.

java.lang.String	**getDispositivo**(java.lang.String login, java.lang.String tipoUsuario) Método usado por outros Serviços do pBuy para obter o dispositivo do usuário.
UsuarioComum	**getUsuario**(java.lang.String id) Responsável por retornar o objeto com as informações do usuário, sendo usado pelos outros Serviços do pBuy.
java.lang.String	**logaUsuario**(java.lang.String login, java.lang.String senha, java.lang.String tipoUsuario, java.lang.String dispositivo, java.lang.String ip) Responsável por logar o usuário na base de dados
static void	**main**(java.lang.String[] args)
java.lang.String	**mostraUsuariosAtivos**() Mostra a lista de usuários ativos no sistema.
java.lang.String	**setStatus**(java.lang.String id) Responsável por setar o status do usuário
void	**trataTimeout**(java.lang.String id) Método que será executado quando estourar o tempo de validade do Ticket, e então vai mover o usuário para os inativos.
java.lang.String	**verificaSeAtivo**(java.lang.String nome, java.lang.String tipoUsuario) Responsável por verificar se o usuário esta ativo ou não sendo utilizado pelo Serviço de Autenticação baseado em Credenciais
UsuarioComum	**verificaUsuario**(java.lang.String ticket) Responsável por verificar se o usuário esta ativo.

Methods inherited from class java.rmi.server.UnicastRemoteObject

clone, exportObject, exportObject, exportObject, unexportObject

Methods inherited from class java.rmi.server.RemoteServer

getClientHost, getLog, setLog

Methods inherited from class java.rmi.server.RemoteObject

equals, getRef, hashCode, toString, toStub

Methods inherited from class java.lang.Object

finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

Autenticador

```
public Autenticador()
            throws java.rmi.RemoteException
```

Método construtor da classe Autenticador

Throws:

`java.rmi.RemoteException`

Method Detail

main

```
public static void main(java.lang.String[] args)
```

---

logaUsuario

```
public java.lang.String logaUsuario(java.lang.String login,
                                    java.lang.String senha,
                                    java.lang.String tipoUsuario,
                                    java.lang.String dispositivo,
                                    java.lang.String ip)
```

Responsável por logar o usuário na base de dados

Parameters:

`login` - login do usuário

`senha` - senha do usuário

`tipoUsuario` - tipo do usuário que é uma classificação do pBuy

`dispositivo` - dispositivo usado pelo usuário no momento da autenticação

Returns:

String contendo o id (= ticket) ou null se não for possível logar o usuário

---

geraId

```
public java.lang.String geraId()
```

Responsável por gerar o ticket que será o identificador do usuário. O ticket serve para identificar o usuário pois é através deste que o usuário vai provar sua autenticidade para o Autenticador. Este ticket é uma String gerada de forma aleatória.

Returns:

String contendo o ticket gerado para o usuário

---

mostraUsuariosAtivos

```
public java.lang.String mostraUsuariosAtivos()
```

Mostra a lista de usuários ativos no sistema.

Returns:

String contendo os usuários ativos.

---

trataTimeout

```
public void trataTimeout(java.lang.String id)
```

Método que será executado quando estourar o tempo de validade do Ticket, e então vai mover o usuário para os inativos.

Parameters:

`id` - identificador do usuário

---

setStatus

```
public java.lang.String setStatus(java.lang.String id)
```

Responsável por setar o status do usuário

Parameters:

`id` - identificador do usuário

Returns:

String contendo o status do usuário

---

estaAtivo

```
public boolean estaAtivo(java.lang.String login,
                         java.lang.String tipoUsuario)
                  throws java.rmi.RemoteException
```

Método usado pelos outros Serviços da arquitetura para saber se o usuário esta ativo ou não

Parameters:

`login` -

`tipoUsuario` -

Returns:

retorna verdadeiro ou falso

Throws:

`java.rmi.RemoteException`

---

getDispositivo

```
public java.lang.String getDispositivo(java.lang.String login,
                                       java.lang.String tipoUsuario)
                                throws java.rmi.RemoteException
```

Método usado por outros Serviços do pBuy para obter o dispositivo do usuário.

Parameters:

`login` - login do usuário

`tipoUsuario` - tipo do usuário

Returns:

String contendo o dispositivo usado pelo usuário

Throws:

`java.rmi.RemoteException`

---

getUsuario

```
public UsuarioComum getUsuario(java.lang.String id)
                        throws java.rmi.RemoteException
```

Responsável por retornar o objeto com as informações do usuário, sendo usado pelos outros Serviços do pBuy.

Parameters:

`id` - identificador do usuário

Returns:

objeto com as informações do usuário

Throws:

`java.rmi.RemoteException`

---

verificaUsuario

```
public UsuarioComum verificaUsuario(java.lang.String ticket)
                             throws java.rmi.RemoteException
```

Responsável por verificar se o usuário esta ativo.

Parameters:
`ticket` - ticket que identifica o usuário
Returns:
retorna o objeto com as informações do usuário
Throws:
`java.rmi.RemoteException`

---

autenticado

```
public boolean autenticado(boolean t)
```

Responsável por verificar se esta ativo

---

verificaSeAtivo

```
public java.lang.String verificaSeAtivo(java.lang.String nome,
                                         java.lang.String tipoUsuario)
```

Responsável por verificar se o usuário esta ativo ou não sendo utilizado pelo Serviço de Autenticação baseado em Credenciais
Parameters:
`nome` - nome do usuário
`tipoUsuario` - tipo do usuário
Returns:
String no formato "ok+ticket" ou null se o usuário não estar ativo

---

autenticacaoCredenciais

Class recebevCard

```
java.lang.Object
  java.lang.Thread
    autenticacaoCredenciais.recebevCard
```

All Implemented Interfaces:
java.lang.Runnable

```
public class recebevCard
extends java.lang.Thread
```

Classe responsável por abrir o Servidor Socket

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields inherited from class java.lang.Thread

MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

recebevCard()
Construtor da classe recebevCard

Method Summary

void run()

void setAutenticadorExecuta(Autenticador autenticador)
Armazena a classe Autenticador e dispara a thread que abrirá o servidor socket

Methods inherited from class java.lang.Thread

activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

recebevCard

public recebevCard()
Construtor da classe recebevCard

Method Detail

setAutenticadorExecuta

public void setAutenticadorExecuta(Autenticador autenticador)
Armazena a classe Autenticador e dispara a thread que abrirá o servidor socket

Parameters:

autenticador - classe responsável pelas operações de manutenção das informações de usuários ativos e inativos no sistema

run

`public void run()`

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `java.lang.Thread`

---

autenticacaoCredenciais

Class TratavCard

`java.lang.Object`

- java.lang.Thread
  - `autenticacaoCredenciais.TratavCard`

All Implemented Interfaces:

java.lang.Runnable

`public class TratavCard`
`extends java.lang.Thread`

Classe responsável pelo recebimento das mensagens enviadas pelo Serviço de Comunicação

---

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields inherited from class java.lang.Thread

`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary

`TratavCard()`

`TratavCard(java.net.Socket clienteSocket)`

Construtor da classe TratavCard

Method Summary

java.lang.String	formataMensagemResposta(java.lang.String mensagem) Formata a mensagem que vai ser enviada ao usuário
void	run()
void	setAutenticador(Autenticador autentica)
void	setAutenticadorExecuta(Autenticador autentica) Armazena a classe autenticador e executa a thread responsável pelo recebimento das mensagens

Methods inherited from class java.lang.Thread

activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

TratavCard

public **TratavCard**(java.net.Socket clienteSocket)

Construtor da classe TratavCard

Parameters:

clienteSocket - cliente que conectou-se ao servidor socket criado anteriormente. Este enviará as mensagens.

TratavCard

public **TratavCard**()

Method Detail

setAutenticadorExecuta

public void **setAutenticadorExecuta**(Autenticador autentica)

Armazena a classe autenticador e executa a thread responsável pelo recebimento das mensagens

Parameters:

autenticador - classe responsável pelas operações de manutenção das informações de usuários ativos e inativos no sistema

---

setAutenticador

```
public void setAutenticador(Autenticador autentica)
```

run

```
public void run()
```

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `java.lang.Thread`

formataMensagemResposta

```
public java.lang.String formataMensagemResposta(java.lang.String mensagem)
```

Formata a mensagem que vai ser enviada ao usuário

Parameters:

`mensagem` - Mensagem sem a formatação (não possui as flags que sinalizam o fim da mensagem).

Returns:

String contendo a mensagem formatada.

autenticacaoCredenciais

Class manipulaMensagens

```
java.lang.Object
```

autenticacaoCredenciais.manipulaMensagens

```
public class manipulaMensagens
extends java.lang.Object
```

Classe responsável por manipular e compreender as informações contidas no padrão vCard.

Constructor Summary

`manipulaMensagens()`

Construtor da classe manipula Mensagens

Method Summary

`java.lang.String`	`addTag(java.lang.String vCard, java.lang.String nomeTag, java.lang.String conteudoTag)` Estende o vCard adicionando novas credenciais desejadas.
`java.lang.String`	`getConteudoTag(java.lang.String tagvCard, java.lang.String nomeTag, int tipoTag)` Obtém os conteudos das tags armazenadas no padrão vCard.
`java.lang.String`	`getSenha(java.lang.String mensagem)` Obtém a Senha recebida numa mensagem
`java.lang.String`	`getvCard(java.lang.String vCard)`

Obtém o vCard do usuário

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

manipulaMensagens

`public manipulaMensagens()`

Construtor da classe manipula Mensagens

Method Detail

getConteudoTag

```
public java.lang.String getConteudoTag(java.lang.String tagvCard,
                                        java.lang.String nomeTag,
                                        int tipoTag)
```

Obtém os conteudos das tags armazenadas no padrão vCard.

Parameters:

`tagvCard` - vCard contendo as informações do usuário

`nomeTag` - nome da tag que se deseja obter o conteúdo

`tipoTag` - usada para distinguir entre as tags do padrão vCard e as credenciais que foram estendidas para uso da aplicação

Returns:

string com o conteúdo da tag desejada

---

addTag

```
public java.lang.String addTag(java.lang.String vCard,
                               java.lang.String nomeTag,
                               java.lang.String conteudoTag)
```

Estende o vCard adicionando novas credenciais desejadas.

Parameters:

`vCard` - vCard contendo as informações do usuário

`nomeTag` - nome da credencial, tag, que vai ser adicionada

`conteudoTag` - conteúdo da tag a ser acrescentada

Returns:

string contendo o vCard já atualizado com a nova credencial que foi adicionada

---

getvCard

`public java.lang.String getvCard(java.lang.String vCard)`

Obtém o vCard do usuário

Parameters:

`vCard` - mensagem contendo várias informações e também o vCard

Returns:

string contendo somente o vCard

---

getSenha

```
public java.lang.String getSenha(java.lang.String mensagem)
```

Obtém a Senha recebida numa mensagem

Parameters:

`mensagem` - mensagem contendo informações e também a senha

Returns:

string contendo a senha contida na mensagem

---

autenticacaoCredenciais

Class trataInformacoesBD

```
java.lang.Object
  autenticacaoCredenciais.trataInformacoesBD
```

```
public class trataInformacoesBD
extends java.lang.Object
```

Classe responsável pelas gerenciar as operações que são necessárias para a manutenção das informações contidas no Repositório de Credenciais.

Constructor Summary

`trataInformacoesBD()`

Construtor da classe trataInformacoesBD

Method Summary

`java.lang.String`	`autalizavCard(java.lang.String vCard, java.lang.String tagAtualizar)` Classe responsável por atualizar o vCard do usuário no repositório.
`boolean`	`consultavCard(java.lang.String vCard)` Método responsável por verificar se o usuário dono do vCard possui credenciais no repositório.
`boolean`	`inserevCard(java.lang.String vCard)` Classe responsável por inserir um novo vCard no Repositório de Credenciais.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

trataInformacoesBD

```
public trataInformacoesBD()
```

Construtor da classe trataInformacoesBD

Method Detail

autalizavCard

```
public java.lang.String autalizavCard(java.lang.String vCard,
                                       java.lang.String tagAtualizar)
```

Classe responsável por atualizar o vCard do usuário no repositório. Essa classe monta a expressao SQL para a atualização de determinadas credenciais contidas no vCard do usuário.

Parameters:

`vCard` - vCard estendido com as credenciais do usuário

`tagAtualizar` - tag, credencial, que vai ser atualizada no Repositório de Credenciais.

Returns:

String contendo o vCard com a Credencial desejada atualizada

---

inserevCard

```
public boolean inserevCard(java.lang.String vCard)
```

Classe responsável por inserir um novo vCard no Repositório de Credenciais.

Parameters:

`vCard` - vCard contendo as informações do usuário e as credenciais de interesse da aplicação.

Returns:

boolean certificando se o vCard foi inserido (true) ou se não foi (false).

---

consultavCard

```
public boolean consultavCard(java.lang.String vCard)
```

Método responsável por verificar se o usuário dono do vCard possui credenciais no repositório.

Parameters:

`vCard` - vCard do usuário

Returns:

boolean certificando se existe ou não credenciais armazenadas no repositório do usuário dono do vCard

---

autenticacaoCredenciais

Class conectaBanco

```
java.lang.Object
  autenticacaoCredenciais.conectaBanco
```

```
public class conectaBanco
extends java.lang.Object
```

Responsável pelas operações realizadas diretamente com o banco de dados

Constructor Summary

`conectaBanco()`

Construtor da clase conectaBanco

Method Summary

void	`atualizaBanco(java.lang.String expressao)` Método que atualiza o banco de dados
java.lang.String	`consultaBanco(java.lang.String expressao)` Método que realiza consulta ao banco de dados.
void	`fechaBanco()` Método responsável por fechar a conexão com o banco de dados
java.lang.String	`trataResultado(java.sql.ResultSet rs)` Método responsável por tratar o resultado do banco de dados

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

conectaBanco

```
public conectaBanco()
```

Construtor da clase conectaBanco

Method Detail

atualizaBanco

```
public void atualizaBanco(java.lang.String expressao)
                   throws java.sql.SQLException
```

Método que atualiza o banco de dados

Parameters:

`expressao` - expressão SQL para atualizar o banco de dados

Throws:

`java.sql.SQLException`

---

consultaBanco

```
public java.lang.String consultaBanco(java.lang.String expressao)
                               throws java.sql.SQLException
```

Método que realiza consulta ao banco de dados.

Parameters:

`expressao` - expressão SQL para realizar a consulta no banco de dados

Returns:

string contendo o resultado da consulta

Throws:

`java.sql.SQLException`

---

trataResultado

```
public java.lang.String trataResultado(java.sql.ResultSet rs)
                                throws java.sql.SQLException
```

Método responsável por tratar o resultado do banco de dados

Parameters:

`rs` - contém o tipo de dado resultante da consulta ao banco de dados

Returns:

string com as informações já tratadas

Throws:

`java.sql.SQLException`

---

fechaBanco

```
public void fechaBanco()
             throws java.sql.SQLException
```

Método responsável por fechar a conexão com o banco de dados

Throws:

`java.sql.SQLException`
