Sistemas de localização dinâmica de serviços em ambientes de computação móvel

Leonardo M. A. Pinho, Alfredo Goldman
{lmap,gold}@ime.usp.br

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo, Brasil

28 de Setembro de 2003

Resumo

Um dos desafios atuais, com o aumento do número de serviços em uma rede, é a localização dinâmica dos mesmos. Várias tecnologias de localização de serviços foram desenvolvidas para simplificar o uso dos mesmos, permitindo que eles sejam encontrados, configurados e usados com um mínimo de esforço. Entretanto, não existe nenhum sistema que selecione o servidor mais conveniente ao cliente, como o mais próximo a este. Apresentamos uma solução para estender os sistemas existentes, integrando-os com sistemas de localização física de unidades móveis, a fim de localizar o serviço mais próximo ao cliente.

1 Introdução

Uma rede de computadores é um conjunto de subredes e computadores interligados. Ela oferece aos seus usuários um enorme potencial para compartilhamento de recursos, tais como documentos, programas, dados e principalmente serviços.

Definimos serviço como uma coleção de componentes distribuídos colaborando entre si, a fim de disponibilizar um recurso ao cliente, seja ele físico, como uma impressora, ou lógico, como um envio de e-mail. Além desses serviços básicos, estão surgindo novos serviços a cada dia.

Nesse ambiente, mecanismos para descoberta de serviços tornam-se extremamente importantes. Sem eles, os usuários utilizam uma fração muito pequena dos benefícios de uma rede, por falta de conhecimento dos seus recursos.

Outro fator importante para a utilização dos serviços de uma rede é a forma de acesso a estes. Idealmente, os usuários gostariam de acessar os recursos automaticamente, sem a necessidade de reconfigurar o seu sistema, como instalar *drivers* manualmente, ou procurar o endereço IP do servidor.

Assim, é importante que os serviços sejam acessados com o mínimo de esforço, ou seja, com a menor reconfiguração possível no dispositivo do cliente que deseja utilizar o serviço.

1.1 Ambiente Móvel

O cenário se torna mais problemático quando analisamos as redes sem fio e suas respectivas unidades móveis[1] (UMs). Nesse tipo de rede, muito comum ultimamente, os usuários mudam frequentemente e por isso não tem conhecimento dos serviços disponíveis na rede e nem de que forma acessá-los.

Imagine uma pessoa que deseja imprimir um arquivo do seu dispositivo móvel (Palm, IPaq, etc) conectado a uma rede sem fio de um prédio comercial. Na maioria das vezes, o usuário precisa contactar o administrador da rede para descobrir o endereço da impressora mais próxima, suas características e a respectiva forma de acesso.

Seria interessante, se existisse um sistema em que o usuário simplesmente informasse o interesse pelo serviço de impressão, e o sistema automaticamente encontrasse a impressora mais próxima a ele de acordo com a sua

[1] Chamaremos de *unidade móvel* um usuário acompanhado de um dispositivo conectado a uma rede sem fio no padrão IEEE 802.11b.

localização, e ainda, reconfigurasse o seu dispositivo para que fosse possível a utilização da mesma.

Um outro exemplo interessante é uma sala de conferência em que o palestrante possui a sua apresentação em um dispositivo móvel e deseja enviá-la, através da rede sem fio existente, para o projetor da sala, que está conectado à rede física do prédio. Gostaríamos que o usuário acessasse o sistema informando o seu interesse de utilizar o serviço de projeção, e a sua unidade móvel fosse automaticamente reconfigurada para ter acesso transparente ao projetor, sem nenhuma intervenção administrativa. Além disso, seria possível disponibilizar em tempo real o arquivo da apresentação para os outros presentes na palestra portadores de dispositivos móveis.

1.2 Objetivos

Apresentaremos um sistema sensível à localização (*location-aware*) em que o usuário simplesmente informa o interesse por um determinado serviço da rede e a aplicação automaticamente encontra o servidor mais próximo a ele e reconfigura a unidade móvel para que esta acesse o serviço. Utilizaremos como exemplo o serviço de impressão, mas a solução se adequa a qualquer tipo de serviço.

Para desenvolver o sistema, foi necessário solucionar alguns problemas: a-) como prover um mecanismo de busca e acesso transparente aos serviços; b-) como obter a localização das unidades móveis em uma rede sem fio.

Atualmente existem no mercado várias tecnologias que visam resolver o problema a-), são os chamados sistemas de descoberta de serviços (SDS). Na próxima subseção explicaremos em detalhes o funcionamento desses arcabouços (*frameworks*) e as respectivas modificações realizadas.

Na seção 3 explicamos a solução encontrada para o problema b-). Na seção 4, por sua vez, descrevemos o protótipo construído, que integra as soluções apresentadas. E finalmente, na última seção, apresentamos algumas conclusões.

2 SDS

As tecnologias para descoberta de serviços foram desenvolvidas visando reduzir a complexidade da utilização dos serviços, e assim, simplificar o uso de dispositivos móveis em uma rede.

Para isso, essas tecnologias provêem uma localização dinâmica (*on the fly*) dos serviços disponíveis. Alguns sistemas também oferecem mecanismos de divulgação de novos serviços e acesso transparente aos mesmos, atuando como uma "ponte" (*middleware*) entre o usuário e o provedor do serviço. Sistemas mais avançados fornecem ainda um "esqueleto" para a implementação de um serviço, servindo como um arcabouço na criação dos mesmos.

Todas as tecnologias de descoberta de serviços possuem um sistema de catálogo, equivalente a uma lista de classificados ou "páginas amarelas", para que o cliente tenha conhecimento dos serviços disponíveis na rede ou procure por algum serviço específico, através do seu tipo ou propriedade[2].

Essa forma centralizada de armazenar os serviços estimula a integração dos mesmos, criando comunidades espontâneas. Por exemplo, uma câmera ligada à rede envia uma foto para uma impressora.

Por segurança, os sistemas oferecem mecanismos de tolerância a falhas e exclusão automática de serviços inacessíveis, mantendo sua lista de serviços disponíveis sempre atualizada.

Em todos os cenários descritos existem pelo menos dois elementos: o cliente do serviço (C) e o provedor do serviço (SP). Em alguns sistemas, há um elemento intermediário entre as requisições dos clientes e as respostas dos provedores, geralmente chamado de gerenciador de serviços (SM).

O gerenciador de serviços provê aos servidores uma forma de publicação dos seus serviços. Assim, no banco de dados do SM é armazenado um registro contendo as características e forma de acesso ao serviço.

Através desse catálogo o SM disponibiliza aos clientes da rede um mecanismo de consulta aos serviços disponíveis. A busca geralmente pode ser feita pelo tipo de serviço e/ou por suas características, por exemplo impressora colorida, laser, etc. Utilizando esses filtros o SM busca um servidor que atenda a requisição, retornando ao usuário o(s) servidor(es) encontrado(s).

Infelizmente, os SDSs estudados não resolvem totalmente o problema abordado, pois nenhum sistema possui a funcionalidade de escolher o servidor mais próximo ao cliente. Não há nenhuma integração com sistemas de localização de unidades móveis.

Isso ocorre pois os mesmos, em sua maioria, foram desenvolvidos para redes estruturadas, onde os seus elementos são fixos. Assim, foi necessário adaptar os SDSs para o ambiente de computação móvel, em que a localização é muito importante, integrando-os com sistemas de localização de unidades móveis.

Apresentaremos a seguir uma solução que estende o *Service Location Protocol (SLP)* [23], desenvolvido pelo

[2] Esse tipo de busca só está disponível em alguns sistemas.

IETF SrvLoc working group, para se tornar um sistema sensível à localização. Na próxima seção descreveremos as modificações realizadas no Jini [15], tecnologia desenvolvida pela Sun Microsystems.

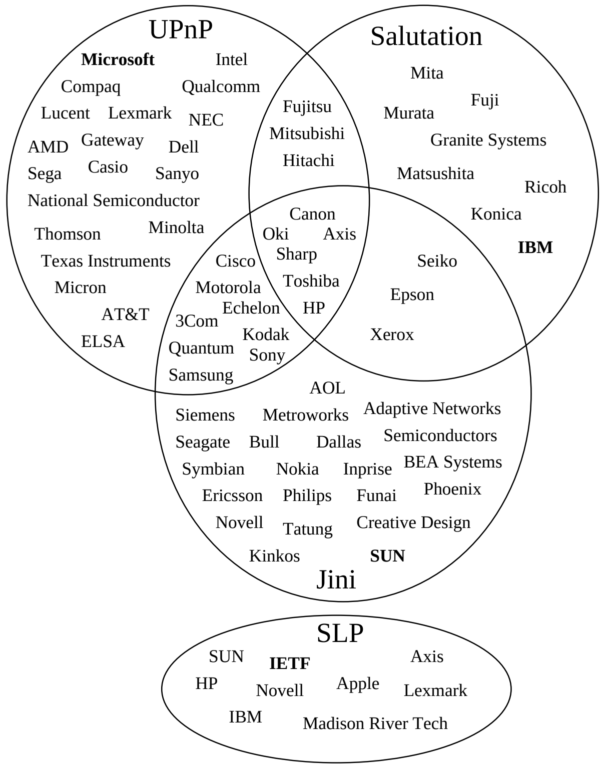

Figura 1: Empresas que desenvolvem SDSs.

Existem outras tecnologias no mercado (ver figura 1), como o *Universal Plug and Play* (UPnP) [3], desenvolvido pela Microsoft; o *Salutation* [3], desenvolvido por um consórcio aberto (*Salutation Consortium*); e o *Bluetooth Service Discovery Protocol* (SDP) [25], que foi criado por um consórcio de empresas de telecomunicações.

Entretanto, devido a algumas limitações dos mesmos, não foi possível testar a solução desenvolvida nesses sistemas. O UPnP não possui um catálogo de serviços, toda a comunicação é feita diretamente entre o cliente e os servidores, não há uma entidade responsável por escolher o melhor serviço ao cliente.

O *Salutation* e o SDP, por sua vez, não provêem suporte a redes sem fio padrão IEEE 802.11b atualmente. A implementação atual do *Salutation* é baseada no protocolo de rede IrDA, embora sua especificação não defina o protocolo a ser utilizado, diferente do SDP, que necessita de uma rede *Bluetooth* para ser utilizado.

2.1 Service Location Protocol (SLP)

Service Location Protocol foi desenvolvido pelo IETF (*Internet Engineering Task Force*) para possibilitar que uma aplicação descubra automaticamente a localização de um serviço desejado, incluindo o endereço (ou domínio) e outras informações de acesso. Como todos os protocolos do IETF, o SLP é especificado com muitos detalhes em documentos denominados *Request For Comments* (RFC).

Em 1997, o *Srvloc working group* do IETF publicou um RFC [16] com a primeira versão do SLP que continha algumas inconsistências. Entretanto, em junho de 1999, o *Internet Engineering Task Force* anunciou a segunda versão do protocolo, adicionando novas funcionalidades, aumentando a segurança e eliminando as inconsistências.

Sua arquitetura é composta por três agentes, que interagem entre si:

- *User Agent* (UA), o usuário do serviço, que procura pela localização de um serviço;
- *Service Agente* (SA), o provedor do serviço, responsável por oferecer um serviço à rede;
- *Directory Agent* (DA), o repositório de serviços, que atua como uma central de informação sobre a localização dos serviços existentes na rede.

A comunicação entre os agentes é feita utilizando 11 tipos de mensagens enviadas através do protocolo de comunicação TCP/IP ou UDP. A mensagem pode ser enviada para um agente específico (*unicast*) ou para todos os agentes interessados (*multicast*).

Todas as mensagens transmitidas são alfa-numéricas precedidas de um cabeçalho binário, como mostra a figura 2. O conteúdo da mensagem é composto por *strings UTF-8* precedidos de um campo informando o tamanho da série de caracteres.

00	Version	Func. ID	Length	
04	Length cont.	Flags		Next extension
08	Next extension continued		XID	
0C	Language tag length		Language tag	

Figura 2: Cabeçalho binário de uma mensagem no SLP.

A comunicação entre os componentes do sistema é feita através de pares de mensagens: `Service Request (SrvRqst) - Service Reply (SrvRply), Service Register (SrvReg) - Service`

`Acknowledgment (SrvAck)`, `Attribute Request (AtrReq) - Attribute Reply (AtrRply)`, por exemplo.

Protocolo

Inicialmente, para que seja feita qualquer ação neste sistema, por parte do SA ou do UA, é necessário descobrir quem é o DA da rede. A figura 3 ilustra o microprotocolo de localização do DA de uma rede.

Os agentes UAs e SAs solicitam a localização de um DA através do envio (via *multicast*) de uma mensagem de requisição do serviço (SrvRqst), procurando por um tipo especial de serviço: `service:directory-agent` (pré-definido pelo protocolo). Quando um DA recebe esta mensagem, ele envia (via *unicast*) para o UA ou SA, uma mensagem `DAAdvert` com o seu endereço (IP) atual.

Obtido o endereço do DA, o provedor de serviço consegue publicar o seu serviço. Para isso, ele executa o microprotocolo ilustrado na figura 3. O SA envia uma mensagem `SrvReg` com as seguintes informações: *Service URL*, tipo de serviço e lista de atributos que descrevem o mesmo.

No SLP, a localização de um serviço é descrita através da *Service URL*[3], que é composta por: endereço IP, número da porta e, dependendo do tipo do serviço, o caminho (*path*). Os clientes (UAs) que obtém essa URL, também chamada de SAP (*Service Access Point*), tem toda informação necessária para se conectar ao serviço.

As características de um tipo de serviço, por sua vez, são descritas através de uma lista de atributos, definidos pelo par nome/valor. Para os tipos de serviços conhecidos existe um documento chamado *Service Templates* [4], registrado na IANA (*Internet Assigned Numbers Authority*), que especifica os atributos e seus valores-padrão para um determinado tipo de serviço. Os SAs utilizam os atributos especificados para descrever o seu serviço, e os UAs os utilizam para procurar um serviço com determinadas características. Isso assegura que os agentes utilizarão o mesmo vocabulário durante essas operações.

Quando o registro é feito com sucesso, o gerenciador de serviços envia uma mensagem `SrvAck` informando que a publicação foi efetuada.

Com isso, o servidor fica disponível na rede, podendo ser consultado. Isso é feito através do envio da mensagem (*unicast*) de `SrvRqst` ao DA, informando o tipo de serviço procurado e, opcionalmente, as características (atributos) que o serviço deve ter.

Quando o DA recebe uma mensagem de `SrvRqst`, que não seja do tipo especial (pré-definido pelo protocolo) `service:directory-agent`, ele realiza uma busca no seu repositório por serviços que atendam à requisição. Em seguida, o DA envia uma mensagem (*unicast*) `SrvRply` contendo todas as URLs dos SAs que atendem ao pedido. Não há nenhuma seleção do servidor mais conveniente ao cliente.

No momento em que o UA recebe as URLs dos SAs, ele obtém a informação necessária para se conectar ao servidor. Entretanto, o protocolo utilizado entre o cliente e o servidor está fora do escopo da especificação do SLP.

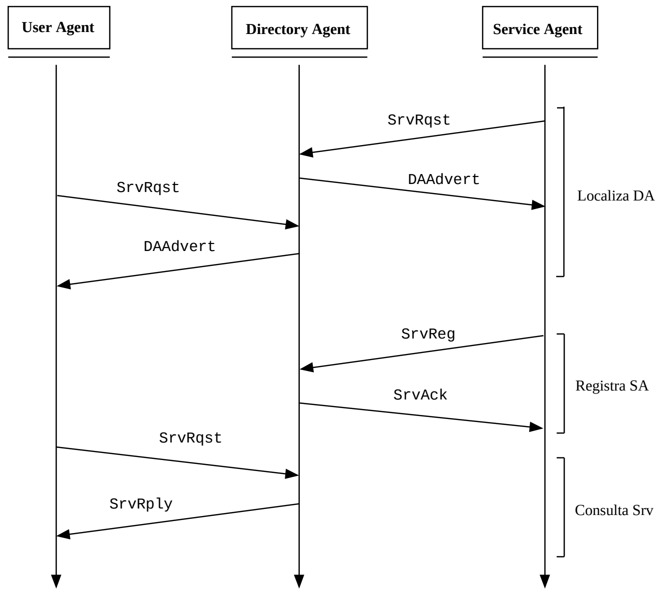

Figura 3: Troca de mensagens no SLP.

Modificações

Para solucionar o problema abordado, foi necessário realizar algumas modificações no SLP, para que o mesmo contemplasse a seleção do servidor mais próximo ao cliente de acordo com a sua localização.

Para isso, o *user agent* precisa informar ao DA a sua localização no momento da requisição do serviço (na próxima seção será mostrado como a UM obtém a sua localização). Sendo assim, foi necessário alterar ligeiramente a especificação do SLP.

Quando o cliente requisita um serviço, este envia uma mensagem `SrvRqst` ao *directory agent* com o tipo de serviço desejado, grupo do servidor, etc. Assim, incluímos um novo tipo de mensagem `SrvRqstWithLocation`, que contém os mesmos campos da mensagem `SrvRqst` mais um campo adicional *User Location*, que informa a localização atual da unidade móvel. Portanto, no momento da requisição de um

[3] URL é abreviação de *Universal Resource Locators*.

serviço, no caso de impressão, o DA terá conhecimento da localização do usuário e poderá encontrar o servidor mais próximo ao UA.

Para disponibilizar o serviço de impressão no SLP utilizamos a implementação da Universidade de Columbia, feita em Java, que possui licença *open source*, podendo ser modificada e redistribuída conforme a necessidade.

Implementamos um *service agent*, que localiza o DA da rede e publica o seu serviço. A publicação do serviço foi feita conforme a especificação do SLP. Entretanto, para que seja possível a busca pelo servidor mais próximo ao cliente, deve existir algum mecanismo em que o SA informa a localização, geralmente fixa, da impressora.

Para solucionar esse problema, criamos um atributo padrão *SrvLocation* que contém a localização do SA no momento do registro do serviço, e poderá ser utilizado pelo DA para decidir se o mesmo está ou não próximo ao cliente.

Após a publicação da URL do serviço, é tarefa do SA iniciá-lo propriamente. Para isso, implementamos uma classe `PrintService`, que é responsável por disponibilizar o canal de comunicação final com a impressora. Por esse motivo, o computador em que ele estiver sendo executado deve conter todo o software (*drivers*, clientes, etc) e *hardware* (acesso físico: cabo paralelo ou rede) necessários para a comunicação com a mesma.

O aplicativo é inicializado fornecendo uma porta e um nome para a impressora. Com esses dados, o sistema abre um canal de comunicação na porta especificada. A partir desse momento, o cliente pode enviar, via *socket*, o arquivo serializado a ser impresso, e o aplicativo reenvia o mesmo para a impressora.

Assim, depois de registrar o serviço, o SA instancia a classe `PrintService` fornecendo como parâmetro a respectiva porta do serviço (registrada no DA). Depois disto, o agente estará pronto para receber requisições na URL publicada.

Além disso, foi necessário modificar o DA para receber mensagens do tipo `SrvRqstWithLocation` e, ao recebê-la, realizar uma busca na sua lista de serviços considerando a localização informada pelo usuário.

Na rotina de seleção do SA adicionamos um novo parâmetro *userLocation* que, após a busca dos servidores, escolhe o SA mais próximo ao cliente e envia as informações de acesso ao *user agent*, que pode enfim utilizar o serviço.

2.2 Jini Network Technology

O *Jini* [14] foi desenvolvido pela Sun Microsystems Inc. com o objetivo de possibilitar a criação de um centro de serviços em uma rede. Através desse elemento, os usuários podem descobrir facilmente os serviços existentes na rede.

Além de localizar o serviço, o Jini também fornece uma forma de acesso transparente ao recurso desejado. Baseado na mobilidade de código do serviço, ele substitui a necessidade de pré-instalação de *drivers* no cliente, automatizando o processo de configuração para acesso a um serviço.

O Jini foi desenvolvido sobre a linguagem Java, utilizando sua tecnologia para objetos distribuídos *RMI - Remote Method Invocation* na comunicação entre os elementos do sistema. Sua implementação atual é baseada nos protocolos de comunicação TCP/IP e UDP, mas implementações baseadas em outros protocolos são possíveis.

O Jini é composto por três elementos: *Lookup Service* (LS), *Service Provider* (SP), *Client* (C). Os dois últimos elementos são os usuários finais do sistema.

O *Service Provider* é o elemento que disponibiliza o serviço e o *Client* é o elemento que deseja utilizá-lo. O *Lookup Service* é um elemento intermediário entre as requisições dos clientes e dos provedores, geralmente chamado de localizador de serviços ou gerenciador de serviços. Ele é o elemento principal do sistema, responsável pelas funcionalidades principais do Jini: mantém a lista dos serviços registrados, gerencia o acesso a um serviço, notifica clientes sobre novos serviços, etc.

Protocolo

No Jini, todas as operações são realizadas através do LS. Portanto, para que um cliente possa acessar um serviço, ou que um servidor possa publicar um serviço, eles precisam inicialmente ter acesso ao LS da rede em que estão conectados.

Isso é feito através da classe `LookupDiscovery` disponibilizada pela API do Jini. O processo de descoberta do LS culmina com a transferência de um objeto remoto (*stub* RMI) referente ao LS específico. O objeto é uma instância da classe `ServiceRegistrar` que possibilita todas as operações sobre o LS encontrado, como a consulta e o registro de serviços. Todas as interações entre o LS e os outros elementos são feitas através do RMI.

Cada provedor de serviço, depois de localizar o LS, deve registrar-se ao mesmo com as informações relevantes à sua descrição, como o tipo de serviço oferecido e, opcionalmente, a lista de atributos que descrevem o serviço, além das informações necessárias para o seu acesso.

No Jini, todas essas informações de acesso ao serviço ficam armazenadas em um objeto denominado *service proxy*, que encapsula a localização do serviço e o protocolo necessário para operá-lo, permitindo um acesso transparente ao recurso. A definição desse objeto é bem flexível, pode ser um *proxy* que contém a informação de conexão ao servidor, como o IP, porta de acesso e protocolo utilizado; ou um objeto remoto, por exemplo um *stub* RMI. A única exigência é que esse objeto seja serializável para ser possível o seu envio ao cliente.

Para o registro efetivo do serviço é utilizado um objeto da classe `ServiceItem`, que contém o *service proxy* e vetor de atributos que descrevem o serviço.

O serviço é publicado através do método `register()` do objeto `ServiceRegistrar`, que recebe como parâmetro uma instância da classe `ServiceItem` e o tempo de expiração desse registro (*leasing time*). O resultado da publicação será armazenado no objeto de retorno `ServiceRegistration`, que contém um *id* identificando o seu registro. O processo é ilustrado na figura 4.

Para uma posterior busca desse serviço é necessário instanciar um objeto que representará a consulta, no caso do Jini, implementado através da classe `ServiceTemplate`. Este objeto é instanciado recebendo três argumentos: *id*, o identificador do serviço, que é opcional e pode ser *null*, o vetor de atributos desejados, e o último argumento, que representa o tipo do serviço, identificado pelo nome de uma *interface* Java. Assim, o LS irá buscar um servidor cujo *service proxy* implemente esta *interface*.

Para efetuar a busca, os clientes utilizam o método `lookup()` de uma instância do *Lookup Service* (`ServiceRegistrar`). Esse método recebe um objeto da classe `ServiceTemplate` como parâmetro e retorna um objeto `ServiceMatches` que contém um vetor de instâncias `ServiceItem`, representando a lista dos servidores encontrados.

Quando o cliente obtém o objeto `ServiceItem`, ele recupera a implementação do serviço (*service proxy*) através da propriedade `service`, e utiliza os métodos disponíveis para ter acesso ao mesmo.

Modificações

Assim como o SLP, o Jini não fornece um método para seleção de serviços depois de encontrado o conjunto de servidores que atendem a requisição. O ideal seria que o sistema automaticamente escolhesse o servidor mais próximo ao cliente. Para disponibilizar essa nova funcionalidade foi necessário modificar alguns procedimentos do Jini.

Para que o cliente pudesse informar ao LS a sua localização, no momento da requisição do serviço, alteramos a especificação do Jini criando uma nova classe denominada `ServiceRegistrarWithLocation`, que estende a

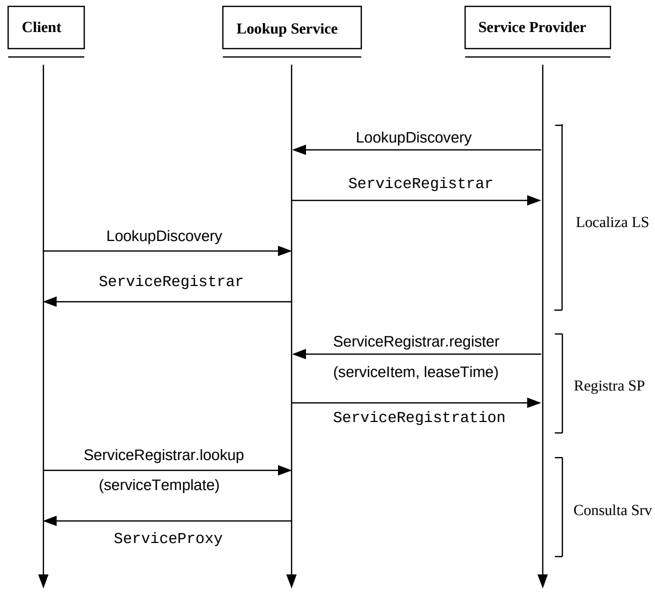

Figura 4: Operações no Jini.

classe `ServiceRegistrar` (representa o *Lookup Service* no Jini) do pacote `net.jini.core.lookup`, sobrecarregando o método `lookup` com um parâmetro adicional *myLocation* que representa a localização do cliente.

Assim, no momento em que o cliente realizar a requisição do serviço, ele poderá fornecer a sua localização atual através do novo método.

Disponibilizamos o serviço de impressão no Jini, implementando o *service provider*, que localiza o LS da rede, publica o seu serviço e inicia o mesmo.

A publicação do serviço foi feita conforme o procedimento padrão do Jini com apenas uma modificação: o SP precisa informar a localização, geralmente fixa, da impressora. Para isto, foi criado um atributo de localização que descreve o serviço no Jini.

O atributo de localização foi implementado na classe `AttrServiceLocation`, que estende a classe `AbstractEntry` para que seja possível a sua inclusão no vetor de atributos que descrevem o serviço, representados no Jini pelo objeto `Entry`. A classe contém um construtor em que é informada a localização do servidor. Assim, o LS pode consultar o atributo de localização do servidor e decidir se o mesmo está próximo ao cliente.

Além disso, conforme a especificação do Jini, é necessário definir a *interface* do serviço, representada pela classe `ServiceInterface`, que contém todos os métodos visíveis (públicos) ao cliente. No nosso caso, foram definidos dois métodos `print` sobrecarregado, que são utilizados para impressão de um arquivo ou de um conjunto de caracteres especificado.

Em conjunto com a *interface* foi criada a classe de *proxy* do serviço (`ServiceProviderProxy`) que implementa a mesma. Nessa classe estão todos os parâmetros do servidor: endereço IP, porta, nome, etc., e o protocolo de comunicação com o mesmo.

Após a publicação da URL do serviço, o SP inicia o serviço propriamente. De forma análoga ao sistema SLP, isto é feito através da instanciação da classe `PrintService` recebendo como parâmetro a porta e nome do serviço (definidos no `ServiceProviderProxy`). Depois disso, o servidor está pronto para receber requisições de algum cliente através do seu *proxy*.

O cliente do serviço foi implementado de forma semelhante a especificação do Jini. A única modificação necessária foi a alteração do objeto responsável pela procura do serviço. Ao invés do cliente instanciar a classe `ServiceRegistrar`, ele instancia a nova classe criada `ServiceRegistrarWithLocation`, que é responsável por encontrar o serviço mais próximo ao cliente, consultando o atributo de localização de cada servidor registrado.

O método `lookup` é chamado informando a localização atual do cliente. O método retorna um objeto serializado `ServiceProviderProxy`, que é usado pelo cliente para utilizar efetivamente o serviço.

3 Sistemas de localização de UMs

Para que o cliente informe a sua localização ao SDS é necessário algum mecanismo que calcule automaticamente a localização atual da unidade móvel.

Com o aumento do número de redes sem fio e de dispositivos móveis (*notebooks*, PDAs, celulares) surgiu a necessidade de desenvolver aplicativos que reagissem com a localização do usuário, comportando-se de forma diferente conforme o contexto em que a unidade móvel se encontra.

Isto é chamado computação sensível a contexto pois, as aplicações assim denominadas, adaptam-se (são sensíveis) às alterações nas condições do ambiente de execução. Definimos "contexto"como qualquer informação que possa ser usada para caracterizar a situação de uma entidade, no nosso caso, a localidade.

Quando o contexto é baseado na localidade, as aplicações móveis também são chamadas aplicações sensíveis à localização (*location-aware applications*). Elas fornecem informações baseadas na localização da UM, como o restaurante mais próximo, hospitais na vizinhança, etc.

Um exemplo típico desse tipo de aplicação é um guia turístico virtual. Existem museus [9] cujo sistema permite que o visitante, portando um dispositivo móvel e conectado a rede sem fio do local, obtenha informações sobre a obra visitada em tempo real. Além disso, poderia ser utilizada uma tecnologia *push*, em que o usuário não precisaria nem requisitar a informação, conforme o seu deslocamento, as informações exibidas na tela seriam atualizadas de acordo com a obra mais próxima a este. Há inúmeras outras funcionalidades que podem ser feitas, como a exibição dinâmica de mapas do museu, controle de obras mais visitadas, trajetos mais comuns na visita ao museu, entre outras.

Esse tipo de aplicação também é utilizada em supermercados cujo carrinho contém um display e na medida que conforme o mesmo mude de corredor, o display do carrinho fornece informações específicas sobre os produtos do corredor atual. O sistema também pode ser instalado em outros tipos de estabelecimentos públicos, como shoppings, feiras de convenções, entre outros.

Entretanto, para que seja possível o desenvolvimento de aplicações sensíveis à localização é necessário um sistema que consiga rastrear e fornecer a localização de uma unidade móvel. A partir da década de 90, começaram a surgir sistemas com essa finalidade.

As principais tecnologias foram desenvolvidas para ambientes *indoor*, tais como ParcTab [10], Active Badges [27], Ekahau [6], Microsoft Radar [21], entre outras. Posteriormente surgiram sistemas para ambientes externos como o GPS [28].

As tecnologias também se diferenciam em relação ao tipo de localização que fornecem, simbólica ou absoluta, precisão e principalmente, às suas necessidades. A maioria dos sistemas necessita de *hardware* proprietário, que é acoplado à unidade móvel, e em alguns casos, de uma rede específica para o sistema.

Para solucionar o problema proposto, gostaríamos que a tecnologia de localização não utilizasse nenhum *hardware* adicional, apenas a rede sem fio para localizar a unidade móvel. Por esse motivo, as opções se restringiram ao Ekahau [13] ou o Microsoft Radar [1].

O sistema foi desenvolvido utilizando o Ekahau. Entretanto, a implementação foi feita de forma a tornar o sistema flexível quanto a tecnologia de localização utilizada. Criamos uma *interface* (`LocationInterface`) que define o esqueleto das classes que fornecem a localização do dispositivo móvel. Para utilizar um outro de sistema de localização, basta criar uma classe que implemente a `LocationInterface`.

3.1 Ekahau Positioning Engine

O sistema de localização Ekahau é um aplicativo comercial desenvolvido por uma empresa da Finlândia e é muito utilizado na Europa. Algumas universidades, como a de Copenhagem (Dinamarca) [6], mantém laboratórios para o desenvolvimento de aplicativos sensíveis à localização (*location-aware*) utilizando o Ekahau.

A principal razão para o sucesso desse sistema é o seu processo de localização, que é feito totalmente via *software*. Diferente de outros sistemas, ele não requer nenhum *hardware* proprietário adicional, pois o mesmo utiliza apenas a infra-estrutura da rede sem fio, obrigatoriamente do padrão IEEE 802.11b, para efetuar a localização de um dispositivo conectado à rede. Obviamente, ele é um sistema de localização *indoor*.

A localização pode ser fornecida de forma simbólica e/ou absoluta. Os tipos de retorno na forma simbólica são definidos pelo cliente através de áreas lógicas em um mapa. Posteriormente mostraremos como criá-las. A precisão da localização fornecida é variável de acordo com alguns parâmetros: número e disposição das estações rádio-base (ERBs), configuração do Ekahau, etc., mas estima-se que o sistema tenha em média 3 metros de precisão em 75% das medições.

Sua heurística principal se baseia na diferença da intensidade de sinal (*Received Signal Strength Intensity* - RSSI) entre locais físicos diferentes. No processo de instalação do Ekahau é realizado um processo de calibragem do sistema (explicado na próxima seção) em que é armazenada a intensidade do sinal em alguns pontos do ambiente coberto pela ERBs. Esses pontos são chamados de pontos de amostra (*sample points*).

Baseado nos pontos de amostra armazenados, o Ekahau consegue comparar a intensidade de sinal da UM com os mesmos e inferir a localização do dispositivo.

O módulo responsável por realizar esse processo é chamado Ekahau Positioning Engine. Ele foi implementado em Java e pode ser acessado através de uma API Java disponibilizada pelo sistema. Além deste módulo, a arquitetura é composta por outros quatro módulos:

- *Positioning Engine*, aplicativo Java, disponibiliza a localização e rastreamento de UMs aos clientes (Windows 2k/XP);
- *Manager*, analisa a precisão do sistema, áreas cobertas e gerencia áreas lógicas (Windows 2k/XP);
- *Calibrator*, responsável por gravar os dados da calibragem da área para a *Positioning Engine* (Windows 2k/XP e CE);
- *Device Service*, serviço executado nas UMs a serem rastreadas (Windows 2k/XP e CE).

O processo de calibragem do sistema é feito através do *Ekahau Calibrator*. Através dele, o usuário poderá armazenar os pontos de amostras em um arquivo de dados (.ecd) para posteriormente serem lidos pela *Positioning Engine*.

O aplicativo permite ainda o cadastro de um mapa (.bmp ou .png) com a visualização do ambiente a ser rastreado. Isso permite a visualização no mapa dos pontos de amostra.

Há um procedimento para criar os pontos de amostras. O usuário precisa se fixar em um ponto do ambiente, tentando não se mover, e girar lentamente 360^o em volta do seu próprio eixo, simulando a rotação do dispositivo. Esse procedimento deve ser repetido para cada ponto de amostra a ser armazenado.

Para melhorar ainda mais a precisão do sistema, existe uma técnica para a escolha dos pontos de amostras. Estes devem ser escolhidos a cada 5-20m^2 dependendo do tamanho do ambiente. Os pontos também não devem estar próximos a paredes ou outros objetos que interfiram no sinal da ERB.

Em cada ponto de amostra, o *Ekahau Calibrator* armazena a intensidade de sinal (RSSI) em decibéis (dBm) de cada canal da ERB.

Para áreas problemáticas com pouco sinal ou com muita variação de sinal, é recomendado gravar mais pontos de amostra. A identificação dessas áreas pode ser feita pelo *Ekahau Manager*, que é o módulo responsável por diagnosticar problemas com o sistema de localização. Ele analisa a precisão do sistema, a coleta dos pontos de amostra, entre outros dados.

Além disso, o aplicativo permite a criação ou edição de áreas lógicas. O usuário, através de um editor de mapa, identifica cada uma das áreas lógicas. As coordenadas das áreas são armazenadas em um arquivo de dados (.txt) que posteriormente pode ser lido pela *Positioning Engine*.

O *Ekahau Positioning Engine* é uma aplicação do servidor feita em Java que fornece a localização de um dispositivo móvel. Além disso, ele provê uma biblioteca em Java para disponibilizar um mecanismo de requisição ao mesmo.

As aplicações podem utilizar a *Positioning Engine* API para rastrear usuários ou recursos conectados à rede sem fio. A *engine* retornará para o cliente um objeto `TrackedDevice` que contém as informações de localização do mesmo, coordenadas, áreas lógicas, indicador de qualidade da informação de localização, etc.

Para iniciar o módulo é necessário fornecer a porta de requisição que será utilizada, o respectivo arquivo de dados (.ecd) e o arquivo de log.

Com o aplicativo iniciado é possível implementar um cliente que se comunique à aplicação e requisite a localização de dispositivo especificado. Isso é feito através da instanciação da classe `PositioningEngineClient` fornecida pela API, cujo construtor recebe como parâmetro um arquivo de propriedades especificando a porta e endereço IP do servidor a ser utilizado.

Para requisitar a localização de um dispositivo é necessário criar um objeto `TrackedDevice` através do método `createTrackedDevice`, que recebe o endereço IP da UM como parâmetro.

Com o objeto criado, é possível associar uma classe receptora da informação da localização. Esta deve implementar a *interface* `LocationHandler` para que a *Posiotioning Engine* chame o método `handleLocation` informando a localização do dispositivo especificado.

Se o cliente quiser a localização de forma simbólica, utilizando áreas lógicas, ele deve fornecer o arquivo de dados (.txt) que delimita as mesmas. Com isso a *engine* consegue analisar a localização da UM encontrada, comparar com as coordenadas das áreas lógicas fornecidas e devolver para o cliente, através do método `getAreaProbabilities`, uma ou mais áreas lógicas em que a UM possa estar, com a respectiva probabilidade.

O último módulo, *Ekahau Device Service* é o único que não tem intervenção do usuário. É um serviço que deve ser executado em cada UM que pode ser rastreada.

Ele é responsável por informar à *Posiotioning Engine* a intensidade de sinal da unidade móvel. Por isso, depende da placa sem fio utilizada. Atualmente, o ekahau contempla cinco modelos diferentes de placas: Compaq WL110, Nokia, Enterasys RoamAbout, Avaya World Card e Orinoco Gold/Silver.

4 Protótipo

Para testar efetivamente o sistema implementamos um cliente para dispositivos móveis, integrando as soluções já apresentadas. A implementação do protótipo pode ser encontrada em *http://www.ime.usp.br/~lmap/prototipo/index.html*.

O protótipo foi feito totalmente em Java para manter a compatibilidade entre os módulos e aumentar o número de dispositivos móveis compatíveis ao sistema. Utilizamos para testes um Palm m505 e um Ipaq contendo uma placa sem fio (*wireless*). Na versão para o Palm a localização é fornecida pelo usuário. No Ipaq, por sua vez, a localização da UM é obtido através do Ekahau.

Para desenvolver o cliente móvel, escolhemos a arquitetura Waba [5], mais especificamente uma extensão da mesma, com algumas melhorias, chamada SuperWaba. O principal motivo para a escolha foi a grande portabilidade do sistema, que contém máquinas virtuais portadas para muitos dispositivos móveis, inclusive para o Windows CE e Palm OS. Além disso, a API já contém algumas funcionalidades prontas para Palm e Ipaq, o que agilizou o desenvolvimento do sistema.

Uma das funcionalidades, específica para o Palm, é o acesso transparente ao banco de dados nativo do PDA, o que possibilita a fácil recuperação dos dados do catálogo de endereços, lista de tarefas, etc. Assim, disponibilizamos ao usuário, através uma interface gráfica, uma forma de escolha dos dados a serem impressos, tais como catálogo de endereços, lista de tarefas, arquivos e-doc, etc.

A arquitetura do protótipo é mostrada na figura 5: a-) Inicialmente, o SP inicia o serviço; b-) Registra o mesmo no catálogo de serviços com a informação de localidade. Em seguida, o cliente utiliza o serviço através do seguinte procedimento: 1-) O cliente obtém a sua localização; 2-) Requisita o serviço de impressão informando sua localização; 3-) O gerenciador de serviços seleciona o SP mais próximo ao cliente e retorna a informação de como acessá-lo; 4-) O cliente acessa o serviço (através de um *proxy* no caso do Jini) e envia os dados a serem impressos; 5-) Os dados finalmente são impressos pelo serviço de impressão.

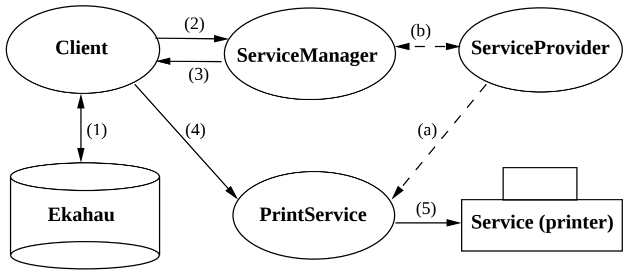

Figura 5: Arquitetura do protótipo desenvolvido.

5 Conclusão

Os sistemas de descoberta de serviço serão muito importantes nos futuros ambientes de rede, como em redes *ad hoc* [2] e redes sem fio, que demandam uma reorganização automática devido a grande mobilidade dos seus elementos.

Eles provêem os serviços da rede aos usuários móveis sem qualquer intervenção administrativa, permitindo um acesso rápido e transparente aos recursos.

Nesses tipos de ambientes, em que os usuários são móveis, é muito importante que as aplicações sejam construídas levando-se em consideração a localização, de forma que ela se adapte ao local em que a unidade móvel se encontra, comportando-se de maneira diferenciada de acordo com a localidade do usuário.

Este artigo mostra como adaptar os SDSs ao ambiente de computação móvel, transformando-os em sistemas sensíveis à localização, que reagem de forma diferente conforme a localização da unidade móvel. Acreditamos que isso incentivará a criação de serviços *location-aware*, como o de impressão, ou o de um guia turístico virtual, entre outros.

A solução foi implementada utilizando dois SDSs distintos: SLP e Jini. O SLP é um protocolo simples, independente de sistema operacional, baseado em troca de mensagens texto, não contempla a forma de acesso ao serviço e demanda poucos recursos dos dispositivos. É indicado para a implementação de serviços simples. O Jini, por sua vez, é um sistema mais completo e flexível, devido a técnica de mobilidade de código utilizada ele permite ao cliente acesso transparente ao serviço. Entretanto, por ser um sistema mais sofisticado, ele demanda mais recursos, o tráfego de rede gerado em suas operações é cerca de 10 vezes maior que o gerado pelo SLP [7].

Referências

[1] P. Bahl and V. N. Padmanabhan. Radar: An In-Bulding RF-based User Location and Tracking System. *Microsoft Research*, March 2000. URL: `<http://research.microsoft.com/bahl/MS_Projects/projects.html>`.

[2] M. Barbeau. Service Discovery Protocols for Ad Hoc Networking. *Carleton University Press*, November 2000.

[3] M. Barbeau. Mobile, Distributed, and Pervasive computing. *Carleton University Press*, 2001.

[4] E. Guttman C. Perkins and J. Kempf. Service Templates and Service: Schemes. *RFC 2165*, June 1999.

[5] C. Catanzaro. An Introduction to Superwaba. *Virginia Tech*, March 2002. URL: <http://csgrad.cs.vt.edu/ccatanza/superWaba.html>.

[6] Anders Cervera. Developing Location Based Applications. Technical report, 2003. URL: <http://positionserver.it-c.dk:8000>.

[7] J. Govea and M. Barbeau. Comparison of Bandwidth Usage: Service Location Protocol and Jini. *Carleton University Press*, 2000.

[8] E. Guttman. Service Location Protocol: Automatic Discovery of IP Network Services. *IEEE Internet Computing*, August:71–80, 1999. URL: <http://computer.org/internet/>.

[9] Forbes Hawkins. Museum Victoria Collection Inventuary System. Technical report, 2003.

[10] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, August:57–66, 2001. URL: <http://computer.org/computer/>.

[11] G. Richard III. Service Advertisement and Discovery: Enabling Universal Device Cooperation. *IEEE Internet Computing*, September/October:18–26, 2000. URL: <http://computer.org/internet/>.

[12] Caldera Systems Inc. An Introduction to SLP. *Whitepaper Caldera Systems, Inc.*, 2001. URL: <http://www.calderasystems.com>.

[13] Ekahau Inc. Ekahau Positioning Engine. Technical report, 2002. URL: <http://www.ekahau.com>.

[14] Sun Microsystems Inc. Jini Architectural Overview. Technical report, 2001. URL: <http://www.sun.com/jini/>.

[15] Sun Microsystems Inc. Jini Network Technology. Technical report, 2001. URL: <http://www.sun.com/jini>.

[16] E. Guttman J. Veizades, C. Perkins and S. Kaplan. Service Location Protocol. *RFC 2615*, June 1997.

[17] A. Loureiro and G. Mateus. *Introdução à Computação Móvel*. Universidade Federal de Minas Gerais - UFMG, 2000.

[18] S. Hodges M. Addlesee, R. Curwen and A. Hopper. Implementing a Sentient Computing System. *IEEE Computer*, August:50–56, 2001. URL: <http://computer.org/computer/>.

[19] D. McCormack M. Barbeau, E. Hughes and F. Bordeleau. Service Recommendation using SLP. *IEEE International Conference on Telecommunications (ICT)*, April 2001. URL: <http://computer.org/internet/>.

[20] R. E. McGrath. Discovery Protocols for Ubiquitous Computing. *University of Illinois Press*, March 2000.

[21] V. N. Padmanabhan P. Bahl and A. Balachandran. Enhancements to the Radar User Location and Tracking System. Technical report, Microsoft Research Technical Report, February 2000. URL: `<http://research.microsoft-.com/bahl/publications.aspx>`.

[22] C. Perkins and E. Guttman. DHCP Options for Service Location Protocol. *RFC 2610*, June 1999.

[23] C. Perkins and E. Guttman. Service Location Protocol, Version 2. *RFC 2608*, June 1999.

[24] C. E. Perkins. Service Location Protocol for Mobile Users. *Sun Microsystems Inc. Press*, 2000.

[25] C. Schwingenschlögl and A. Heigl. Development of a Service Discovery Architecture for the Bluetooh Radio System. *University of Munich Press*, 2000.

[26] J. Waldo. Alive and Well: Jini Technology Today. *IEEE Computer*, June:107–111, 2000. URL: `<http://computer.org/computer/>`.

[27] R. Want and B. Schilit. Expanding the Horizons of Location-Aware Computing. *IEEE Computer*, August:31–34, 2001. URL: `<http://computer-.org/computer/>`.

[28] W. Liao Y. Tseng, S. Wu and C. ChaoY. Location Awareness in Ad Hoc Wireless Mobile Networks. *IEEE Computer*, June:46–52, 2001. URL: `<http:-//computer.org/computer/>`.