UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

DIEGO SANCHEZ GALLO
ERIKA MIDORI KATO
LIN YU CHING

Sistema de Roteamento dentro de um Estabelecimento

São Paulo
2005

DIEGO SANCHEZ GALLO
ERIKA MIDORI KATO
LIN YU CHING

Sistema de Roteamento dentro de um Estabelecimento

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Bacharel em Engenharia da Computação.

Área de concentração:
Computação e Sistemas Digitais

Orientadora:
Profa. Dra. Anna Helena Reali Costa

São Paulo
2005

AGRADECIMENTOS

À Profa. Dra. Anna Helena Reali Costa, pela sua atenção, empenho e dedicação na orientação deste trabalho.

Ao Prof. Dr. Jorge Luis Risco Becerra, pela sua cooperação e aconselhamento nas diversas fases do projeto.

Ao Laboratório Microsoft USP, principalmente aos funcionários Anderson Roberto Sato, Carlos Henrique Esteves Mendonça e Thiago Bierrenbach Carreiro, por sua colaboração e suporte.

Aos nossos pais, namorados e amigos pela compreensão e carinho durante todo este ano.

SUMÁRIO

LISTA DE ABREVIATURAS

BSP: Binary Space Partitioning

CDMA: Code Division Multiple Access

GPS: Global Positioning System

GSM: Global System for Mobile Communications

IEEE: Institute of Eletrical and Eletronics Engineers

JDK: Java Development Kit

JVM: Java Virtual Machine

LAN: Local Area Network

LBS: Location-Based Services

PDA: Portable Device Assistant

RMI: Remote Method Invocation

RPC: Remote Procedure Call

RRT: Rapidly exploring Random Tree

SDK: Software Development Kit

WEB: World Wide Web

Wi-Fi: Wireless Fidelity

WLAN: Wireless Local Area Network

XDR: eXternal Data Representation

XML: eXtensible Markup Language

LISTA DE FIGURAS

RESUMO

O projeto consiste em desenvolver um sistema capaz de auxiliar usuários que possuem restrições de locomoção dentro de estabelecimentos como, por exemplo, shopping centers. Através de seu dispositivo móvel, o usuário acessa o site do sistema, que localiza sua posição por meio de uma rede sem-fio e retorna o melhor caminho a ser percorrido até o local destino solicitado pelo usuário, atendendo inclusive suas possíveis restrições, como a utilização de cadeira de rodas.

As informações sobre o mapa do estabelecimento são estruturadas na forma de grafos hierárquicos, onde cada um de seus níveis apresenta uma abstração mais detalhada do nível imediatamente superior a ele. Tais grafos permitem uma maior eficiência na execução do algoritmo de roteamento, que utiliza o pré-cálculo de alguns caminhos especiais para otimizar o desempenho do sistema.

Quanto à localização do usuário dentro do estabelecimento, tal processo é realizado através da análise do sinal do dispositivo móvel captado pelos pontos de acesso da rede sem-fio. Tendo informado a rota ao usuário, o sistema inicia o monitoramento da trajetória percorrida por ele e, no caso deste desviar-se do caminho indicado, o sistema realiza o replanejamento da rota.

A implementação do projeto foi realizada para o Prédio da Engenharia Elétrica da POLI, e seus resultados experimentais apontam sucessos e dificuldades enfrentados durante este trabalho através da análise dos testes realizados e dos valores atingidos, além de sugestões de melhoria para possíveis implementações futuras.

ABSTRACT

The project consists in developing a system that is capable of helping people who have mobility restrictions in a place such as, for example, a shopping center. Using a mobile device, the user access the site of the system, which locates the user's position through a wireless network, and returns the best path to be followed up to the goal location, respecting possible restrictions, like the need for a wheelchair.

The place's map information is structured into hierarchical graphs, and each level represents a more detailed abstraction than its immediately higher level. These graphs provide more efficiency when running path planner algorithm, which uses pre-paths in order to optimize the system's performance.

Relating to the user's position in the place, this process is realized through the mobile device signal analysis, which is caught by the wireless access points. Once informed the route to the user, the system begins to monitor his path and in case the user deviates from the route received, the system plans the path again.

This project was implemented to Electrical Engineering Building at POLI, and the experimental results obtained show successes and difficulties faced during this work through the analysis of tests and the achieved values, added to suggestions of possible future improvements.

1 INTRODUÇÃO

Atualmente, tanto em estabelecimentos comerciais quanto residenciais, idosos e pessoas portadoras de deficiências físicas enfrentam dificuldades para se locomoverem de um ponto a outro, seja pela falha no desenho dos acessos, seja pela falta de informações objetivas e precisas. Seria interessante que estas pessoas pudessem adquirir informações que identificassem alternativas de caminhos mais adequados às suas dificuldades e destinos.

Esse tipo de serviço tem se tornado cada vez mais comum, podendo ser encontrado nos mais diversos ambientes. Sua disponibilização depende de uma infra-estrutura que permita a comunicação entre dispositivos móveis. A expansão das redes sem fio e a popularização de dispositivos portáteis que acessem essas redes tornam bastante viável a implementação deste serviço, de forma que a pessoa possa consultar, por meio de um dispositivo portátil, a melhor rota a seguir, respeitando suas dificuldades.

1.1 Objetivos

Este trabalho tem por objetivo o desenvolvimento de um sistema capaz de informar, por meio de um dispositivo portátil e a partir da localização do usuário dentro de um estabelecimento, o melhor caminho a ser percorrido para chegar a uma localização alvo, de acordo com as necessidades e possibilidades do usuário.

1.2 Organização do Texto

Este trabalho está organizado em cinco capítulos. O capítulo 2 descreve algumas formas de representação espacial, dados por mapa, e especifica o sistema proposto. O capítulo 3 detalha o projeto e implementação do sistema, cujos resultados experimentais se encontram descritos no capítulo 4. Finalmente, o capítulo 5 apresenta as conclusões as conclusões alcançadas e delineia trabalhos futuros a serem realizados.

2 ESPECIFICAÇÃO DO PROJETO

O sistema de roteamento proposto neste trabalho permite que um usuário dentro de um estabelecimento possa, através de seu PDA, tablet ou notebook (ver Figura 2.1), consultar o melhor caminho, de acordo com suas possibilidades, para chegar a determinado local. O sistema então identifica a localização do usuário através do dispositivo móvel e de uma rede wireless, calcula a rota a ser seguida, e informa a mesma ao usuário. Este sistema central pode atender concorrentemente a várias consultas de usuários.

Por se tratar de um sistema bastante complexo, o mesmo foi dividido nos seguintes módulos:

- ***Representação do mapa*** – Contempla a estrutura de dados que representa o estabelecimento e suas restrições de acesso. Esta representação será utilizada pelo algoritmo de roteamento;
- ***Algoritmo de roteamento*** – O algoritmo indicará o melhor caminho a ser percorrido por um usuário de acordo com suas exigências;
- ***Projeto e Configuração de WLAN*** – Projeto e configuração de uma rede wireless que será utilizada para a comunicação entre dispositivos móveis e o sistema central;
- ***Sistema de localização*** – Este módulo é responsável por informar a posição do usuário dentro do estabelecimento;
- ***Interface com o usuário*** – Este é o módulo que receberá o destino desejado pelo usuário e suas restrições, e apresentará a saída (rota indicada).

Os casos de uso do sistema estão detalhados no Anexo A.

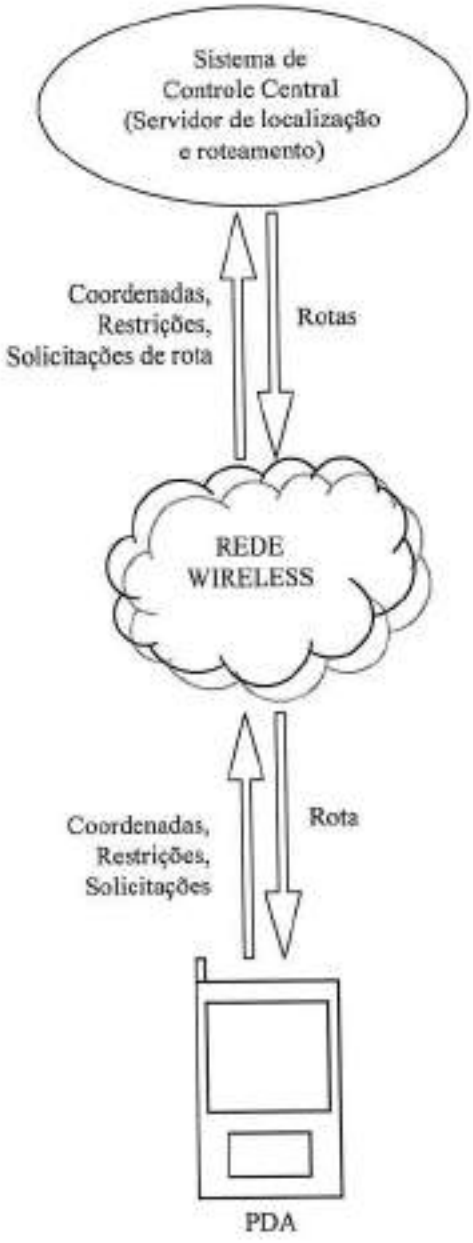


Figura 2.1 – Esquema simplificado do Projeto.

O estabelecimento escolhido para o desenvolvimento deste projeto foi o prédio da Engenharia Elétrica da Escola Politécnica da USP, devido a sua facilidade de acesso e a infra-estrutura disponibilizada.

2.1 Representação do Mapa

A forma mais natural de representar um ambiente é um mapa (DUDEK; JENKIN, 2000). Ele pode incluir diversas informações, como os lugares, as propriedades de reflexão dos objetos, as regiões que são inseguras ou difíceis de atravessar, entre outras.

A representação interna de um ambiente é tipicamente requerida em pelo menos três diferentes classes de tarefas:

1. Para estabelecer quais partes do ambiente estão livres de obstáculos. Esta região é chamada espaço livre.
2. Para reconhecer regiões ou locais do ambiente.
3. Para reconhecer objetos específicos dentro do ambiente.

Na construção de uma representação interna, é preciso escolher quais primitivas serão usadas. Representações baseadas em objetos, entidades simbólicas, ocupação do espaço, lugares, rotas para uma tarefa específica, e informação procedural foram propostas. Em geral, representações espaciais podem ser divididas em dois grupos principais: aquelas que se baseiam em representação métrica e aquelas que são topológicas.

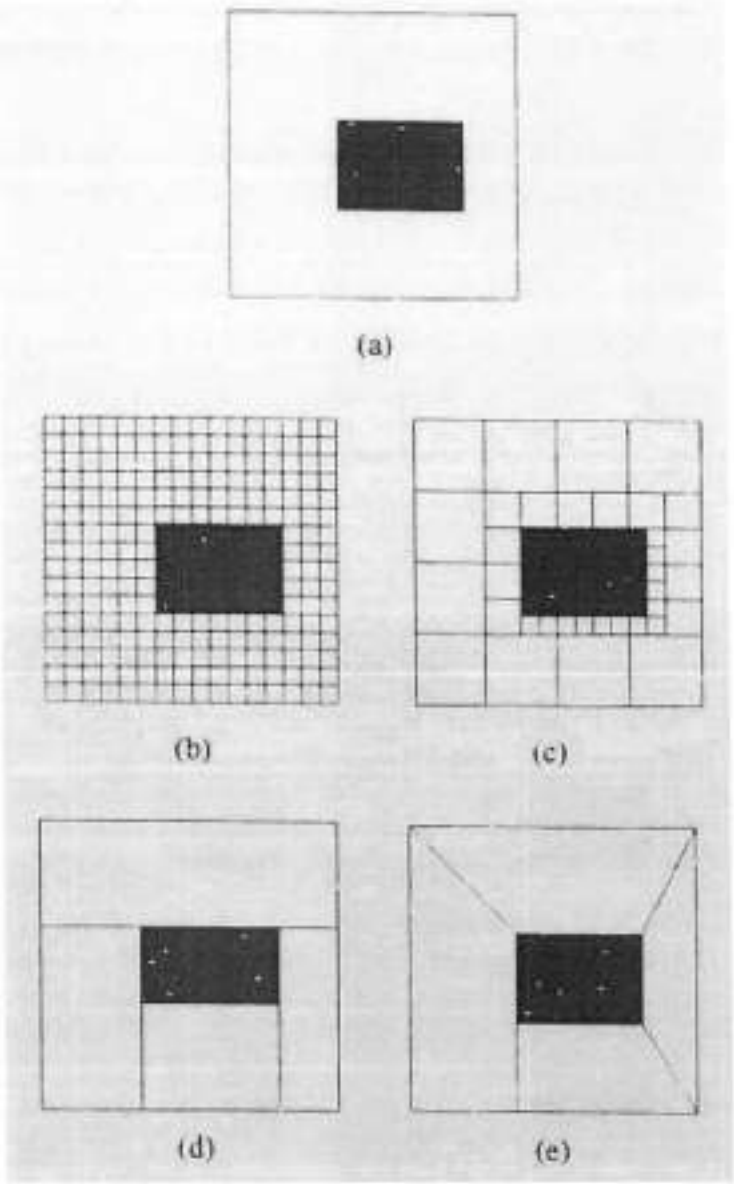
2.1.1 Decomposição Espacial

Talvez a forma mais simples de obter a representação espacial seja amostrar discretamente o ambiente descrito. Tal idéia evita ter que discriminar ou identificar objetos individualmente dentro do espaço estudado. A amostragem pode ser realizada de várias formas, usando um dos diferentes métodos de subdivisão baseados nas formas do objeto, ou mais comumente, definindo uma amostragem através dos nós definidos por uma grade colocada sobre o espaço analisado. O método mais simples utiliza amostragem de células sobre uma grade uniforme.

A principal vantagem da representação com grades regulares é sua extrema generalidade: nenhuma suposição é feita considerando o tipo de objeto. As grades podem representar qualquer coisa. A maior desvantagem deste tipo de representação

é que a resolução ou a fidelidade da grade é limitada pelo tamanho da célula, e que a representação é armazenada ocupando uma grande quantidade de memória mesmo se grande parte do ambiente estivesse vazia ou ocupada.

A Figura 2.2a e a Figura 2.2b mostram um exemplo de ambiente e sua representação baseada em voxels (elementos de amostragem para três dimensões).



(a) Amostra de ambiente;
(b) Decomposição uniforme;
(c) Quadtree;
(d) BSP;
(e) Decomposição exata.

Figura 2.2 – Modelos de Decomposição Espacial (retirada de DUDEK; JENKIN (2000)).

Dada a simplicidade da representação baseada em grade e seus irreais requisitos de armazenagem necessários para representar cada célula da grade explicitamente, uma alternativa é tirar vantagem do fato de muitas das células terem um estado similar, especialmente aquelas que correspondem a regiões adjacentes espacialmente. Uma forma é representar o espaço usando células de formatos e tamanhos não-uniformes, mas uma representação hierárquica recursiva é mais comumente utilizada. O exemplo mais comum é o quadtree.

O quadtree é uma estrutura de dados recursiva para representar uma região quadrada de duas dimensões. É a representação hierárquica que é capaz de reduzir potencialmente o armazenamento e ajudar em certos tipos de computação. O quadtree começa com uma grande região quadrada que inclui todo o espaço necessário. As células que não são nem uniformemente vazias nem cheias são subdivididas em quatro sub-partes iguais. As sub-partes são subdivididas sucessivamente até que sejam uniformemente vazias ou cheias, ou até que uma resolução limite seja atingida. Uma amostra de ambiente e sua representação quadtree são dadas na Figura 2.2a e na Figura 2.2c.

A representação quadtree se torna bastante conveniente para ambientes onde a maior parte do espaço está livre ou ocupado. Entretanto, no pior caso, em que a subdivisão em células mínimas é necessária, o quadtree se torna inadequada devido ao excesso de informações repetidas. Além disso, tal método é bastante dependente do posicionamento dos objetos considerados no ambiente. Dois métodos alternativos de decomposição espacial que não são tão restritivos quanto à representação quadtree são as árvores BSP (Binary Space Partitioning) e o método da decomposição exata.

Uma árvore BSP é uma representação hierárquica na qual uma célula retangular é uma folha ou é dividida em duas árvores BSP por um plano paralelo a uma das fronteiras externas do ambiente. A Figura 2.2d mostra uma possível representação BSP do espaço livre descrito na Figura 2.2a. As árvores BSP fornecem a mesma divisão binária do espaço característica dos quadtrees, porém sem forçar o rigor de ter as mesmas subdivisões.

Uma outra alternativa é subdividir o espaço exatamente em vez de querer uma hierarquia como aquelas introduzidas pelas decomposições quadtrees ou BSP. O

espaço livre é quebrado em regiões não-sobrepostas através de planos tais que a união das partes é exatamente o todo. Um exemplo dessa decomposição pode ser visto na Figura 2.2e. Infelizmente, tanto o método BSP quanto o exato não fornecem representações únicas.

2.1.2 Representações Geométricas

Representações geométricas são aquelas constituídas de primitivas geométricas discretas, como linhas, polígonos ou poliedros, pontos, funções polinomiais, e assim por diante. Tais mapas têm a vantagem de serem altamente eficientes quanto ao espaço, pois uma região arbitrariamente grande pode ser representada por um modelo com apenas alguns parâmetros numéricos. Adicionalmente, mapas geométricos podem armazenar dados de ocupação com resolução quase que arbitrariamente alta sem estarem sujeitos a penalidades incorridas por técnicas baseadas em amostragem espacial.

Modelos geométricos fornecem uma expressão concisa dos dados do ambiente e que pode ser facilmente utilizada em processamento de alto nível. A natureza discreta dos modelos geométricos os faz adequados para interpretações semânticas e julgamentos.

2.1.3 Representações Topológicas

Representações geométricas têm os dados métricos como o centro da representação. Infelizmente, estes dados métricos podem levar a erros de acordo com a forma em que foram obtidos. Para evitar este problema, uma representação topológica não-métrica pode ser usada.

Para representações de espaços de grande escala que são usados por humanos (e outros seres), a topológica parece ser mais adequada que a geométrica.

A chave para a relação topológica é alguma representação explícita de conectividade entre regiões ou objetos, e em sua forma mais pura, ela pode ter completa ausência de dados métricos. Uma representação topológica é baseada em

uma abstração do ambiente em termos de lugares discretos com arestas conectando-os.

Grafos são muito usados para modelar ambientes. Seus vértices correspondem a pontos de referência conhecidos, e suas arestas indicam os caminhos entre eles. Não é necessário um entendimento real da relação geométrica entre os locais em seu ambiente; estes são somente ligados por sua representação topológica. Apesar disso, a representação contém informação suficiente para que um movimento ponto-a-ponto seja conduzido. A representação é também bastante compacta.

2.1.4 Representação de Mapa para o Projeto

Neste projeto, devido à forma como os mapas serão utilizados e ao tipo de usuário do sistema, a representação que se mostrou mais adequada foi a topológica, com a utilização de grafos, com informações métricas a ela associadas.

Grafos são amplamente utilizados para abstrair os ambientes. Entretanto, se o grafo tem uma grande quantidade de nós e arcos, a busca de um caminho e o gerenciamento das informações se tornam um problema de alta complexidade computacional. Conseqüentemente, a informação do grafo precisa ser organizada de forma a reduzir a complexidade e ganhar eficiência e clareza. Uma decomposição hierárquica é necessária e grafos H (CAGIGAS; ABASCAL, 2004) são adequados para este caso.

O grafo H proposto é uma seqüência de níveis hierárquicos, que se diferenciam pelo nível de abstração da representação do ambiente modelado. Em cada nível, há um grafo ponderado formado por conjuntos de nós, arcos, coordenadas cartesianas dos nós, pesos dos arcos, e caminhos pré-calculados associados a um dado nó.

Os nós são classificados em: nós finais (pontos de partida ou destino), nós-cruz (voltas ou cruzamentos), nós submapa (conjunto de nós de um nível de abstração mais profundo) e nós-ponte (conecta um submapa ao seu submapa pai). Os nós-ponte ainda podem ser divididos em duas classes: horizontais (que seguem a definição dada) e verticais (conceitualmente conectam dois submapas que representam dois

andares em um prédio; por exemplo, um elevador). Os diversos tipos de nós podem ser observados na .

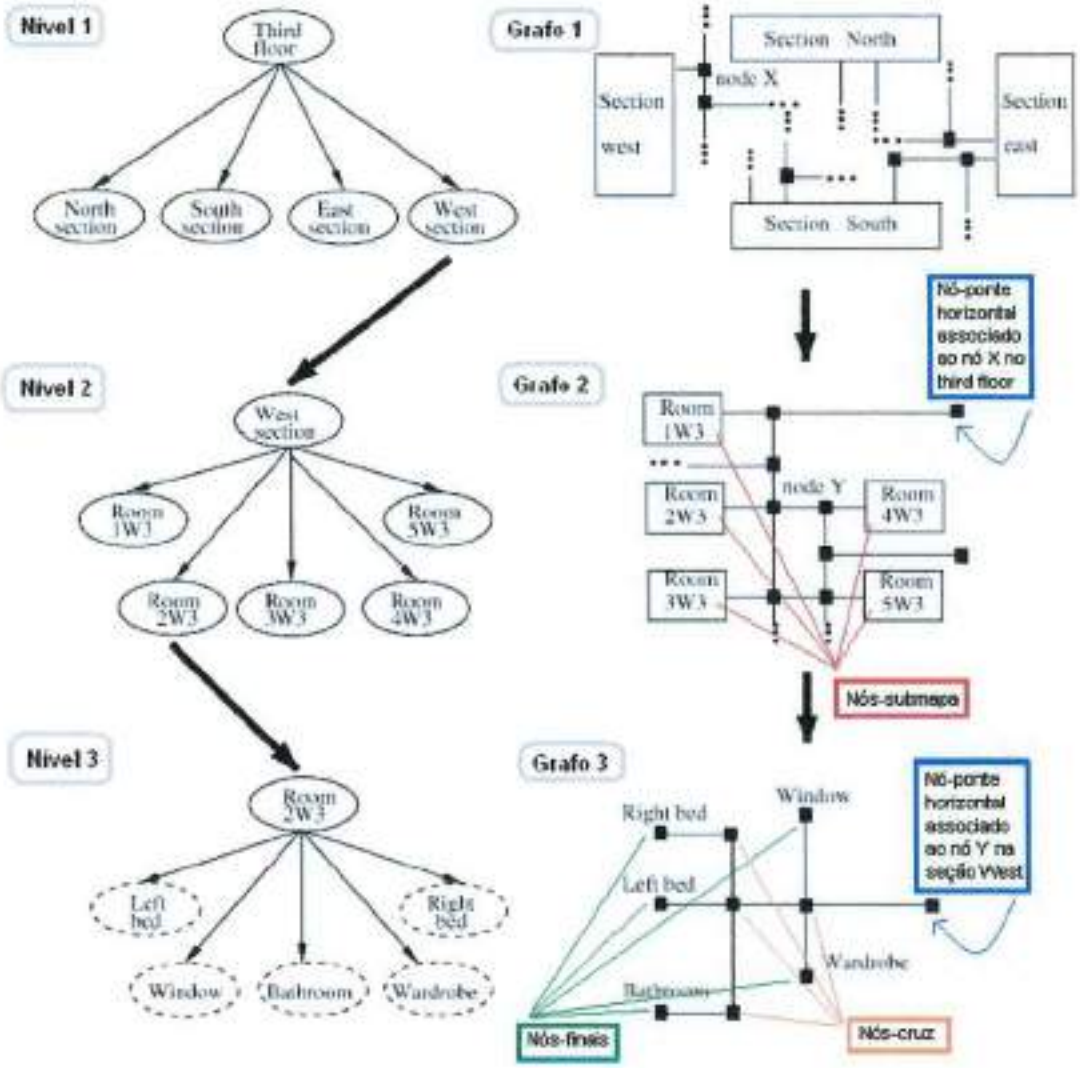


Figura 2.3 – Exemplos dos tipos de nó utilizados.

Os arcos geralmente são não-direcionais e não possuem outros arcos em um nível de abstração mais profundo. O caminho é definido como a sucessão de diversos nós. Um conjunto de caminhos pré-calculados é formado por caminhos de comprimento ótimo que são calculados pelo sistema em uma fase de pré-execução. Eles podem ser agrupados em três classes:

1. Caminhos que ligam dois nós-ponte de um dado nó.
2. Caminhos que ligam nós-ponte de um nó com os nós-ponte de seu submapa pai.
3. Caminhos que ligam submapas "irmãos" contidos em um dado nó.

Estes caminhos evitam muito recálculo no processo de busca hierárquica. Isso é a chamada materialização de custos. Por um lado, ela exige espaço de armazenamento extra para os caminhos e seus custos, além do cálculo prévio. Por outro lado, ela pode garantir otimização com um método clássico de refinamento para busca hierárquica, além de economizar tempo durante a execução do sistema, por evitar o refinamento de nós em níveis de abstração mais profundos da hierarquia.

Adicionalmente, este modelo de mapa é altamente flexível e facilmente adaptável. No caso de necessidade de atualização do mapa, a modularidade intrínseca dos grafos-H faz um fácil recálculo do caminho parcial. Dependendo dos requisitos do módulo de planejamento de caminho, alguns caminhos pré-calculados ou classes de caminhos podem ser adicionados ao grafo-H ou removidos. Este último ponto introduz o conceito de materialização parcial de custos.

2.2 Algoritmo de Roteamento

Utilizaremos um algoritmo de roteamento para realizar o planejamento do caminho, ou seja, a decisão de qual rota tomar baseando-se na representação interna do estabelecimento.

Existem muitos algoritmos de roteamento, mas estes podem ser divididos em três grupos principais: (CHAMPANDARD, 2005):

- *Single Pair:* Trabalham com origem e destino específicos. Somente um caminho é requerido como resposta e normalmente espera-se que seja a solução mais otimizada.
- *Single Source:* Utilizam uma origem e vários destinos. Neste, deseja-se obter o caminho de uma determinada origem para todos os destinos. A

solução é uma lista de (n-1) caminhos, onde n é o número de pontos, e seus respectivos comprimentos, sendo que a forma de representação mais conveniente é uma árvore geradora mínima.

- *All Pairs:* Retorna o caminho mais curto de cada ponto para todos os outros. Este é naturalmente o que tem maior custo computacional.

Para o sistema aqui proposto será utilizado um algoritmo do tipo *single pair*, pois o que se deseja é justamente encontrar o melhor caminho entre uma origem e um destino.

Foram estudadas as vantagens e desvantagens de se utilizar o algoritmo A* (LOPES; POLICARPO; WATT, 2002) e o RRT (Rapidly exploring Random Tree) (KUFFNER; LAVALLE, 2000) para avaliar se estes seriam adequados às necessidades do projeto. O algoritmo A* garante a solução ótima, porém tem complexidade exponencial e poderia consumir muito tempo de processamento, tornando o sistema inviável, enquanto que o RRT foi construído para buscar eficientemente em espaços de configuração de alta dimensionalidade.

Também foram avaliadas algumas variantes destes algoritmos como, por exemplo, o RRT-Connect (KUFFNER; LAVALLE, 2000), que combina o RRT com uma heurística que tenta conectar duas árvores, uma partindo do ponto de origem e outra, do destino. O RTT-Connect foi originalmente desenvolvido para o planejamento de movimentação de um braço humano (modelado como uma cadeia cinemática de sete graus de liberdade).

Devido às restrições em relação ao tempo de resposta do sistema e a necessidade de uma resposta o mais próximo possível da solução ótima, foi decidido utilizar um algoritmo de busca hierárquico, que possibilita uma busca eficiente aliada à resposta rápida. O algoritmo desenvolvido neste projeto teve como base um algoritmo proposto para ser utilizado em cadeiras de rodas inteligentes (CAGIGAS; ABASCAL, 2004), cujo sistema foi denominado TetraNauta.Neste projeto, algumas modificações foram realizadas no algoritmo utilizado pelo TetraNauta. As contribuições realizadas sobre o algoritmo e a representação de Cagigas e Abascal são as seguintes:

- O grafo utilizado tem arcos direcionais, pois esta é uma informação importante para que o sistema decida o melhor caminho, por exemplo, no caso de uma escada rolante, que só permite sua utilização em uma direção.
- Utilização de custos variáveis. Para que o algoritmo possa considerar as restrições de cada tipo de usuário do sistema, optou-se por realizar esta função através da manipulação do custo para "atravessar" um arco, ao invés da utilização de uma representação diferente do mapa para cada tipo de usuário. Com isso, no caso de não existir caminho conveniente, uma alternativa pode ser indicada, mesmo que, neste caso, o usuário tenha dificuldades de locomoção (por exemplo, um paraplégico necessitando subir uma escada). Seria possível até indicar o grau de dificuldade que a pessoa teria para alcançar o destino com base no custo do caminho.
- Outra modificação sobre o algoritmo original é que os caminhos pré-calculados que o algoritmo utiliza precisarão levar em consideração o custo variável. A solução para esta questão foi a de armazenar o melhor caminho para cada tipo de usuário.

2.2.1 Descrição do algoritmo de busca do caminho

Há alguns passos envolvidos no processo de busca do caminho que podem ser resumidos em dois. Primeiro, um conjunto de esqueletos de caminhos que conectam um nó inicial a um nó destino é construído. Segundo, o melhor esqueleto de caminho é completado e retornado. Um esqueleto de caminho não é um caminho completo porque ele não contém uma seqüência contínua de nós entre o nó inicial e o final, ou seja, existem nós intermediários faltantes no esqueleto de caminho que precisam ser adicionados. Os esqueletos de caminho são construídos como uma seqüência de sub-destinos que são estendidos sistematicamente. Estes sub-destinos são nós especiais chamados de nós- pontes.

Basicamente, no processo de desenvolvimento do esqueleto, o algoritmo de busca tenta encontrar os melhores caminhos entre os submapas onde os nós inicial e final estão incluídos. Estes submapas são chamados submapa inicial e submapa

destino, e estão conectados aos seus submapas "pais" em um nível mais alto da hierarquia de abstração através de seus nós-pontes. Este processo é repetido até que ambos os conjuntos de caminhos parciais convirjam num submapa comum. Este último submapa pode inclusive ser o primeiro nível de abstração da hierarquia (submapa raiz no nível 0).

Cada caminho usado para conectar submapas é obtido a partir do conjunto de caminhos pré-calculados que o algoritmo utiliza.

O algoritmo de busca define um processo de busca "bottom-up" ao invés do tradicional processo de refinamento "top-down" tipicamente utilizado quando se trabalha com grafos hierárquicos.

O procedimento de busca consiste em quatro partes que levam em conta quatro diferentes casos. Cada caso dependendo do submapa e do nível da hierarquia em que os vértices inicial e final estão incluídos.

1. Ambos os vértices estão incluídos num mesmo submapa. Este é o caso mais simples, quase equivalente à busca em grafo não-hierárquico e um algoritmo A* é utilizado. Note que se um vértice submapa é encontrado durante o processo de busca, não é necessário seu refinamento devido à existência dos caminhos pré-calculados.
2. Vértice inicial num nível de abstração da hierarquia mais profundo que o vértice final. Um conjunto de esqueletos de caminhos que ligam o vértice inicial ao final é construído.
3. Idem ao anterior com o vértice final num nível de abstração da hierarquia mais profundo que o vértice inicial.
4. Ambos os vértices num mesmo nível de hierarquia. Neste caso, são gerados conjuntos de esqueletos de caminhos para o vértice inicial e para o final até que ambos convirjam num submapa comum num nível superior da hierarquia.

Após estes procedimentos, o melhor esqueleto de caminho é selecionado, completado e retornado.

2.3 *Projeto e Configuração da WLAN*

Neste módulo foi efetuada a construção de uma rede Wireless 802.11 (padrão IEEE) que englobe o sistema de localização e comunicações entre seções e dispositivos móveis dos usuários.

A instalação e a configuração da rede dependem da estrutura do estabelecimento, sendo que para realizar a localização do usuário foram necessários alguns pontos de acesso para capturar os sinais emitidos pelos dispositivos móveis. A configuração consiste em montar a infra-estrutura do estabelecimento de forma a conseguir transformar os sinais capturados em dados utilizáveis para realizar a localização usando um algoritmo de triangulação. (GWON; JAIN; KAWAHARA, 2004); (THAPA; CASE, 2003)

2.4 *Sistema de Localização*

Um sistema de localização atualmente pode ser implementado de diversas maneiras:

- GPS: Global Positioning System

  É um sistema de satélites que analisa o posicionamento de dispositivos receptores terrestres. O GPS se tornou um sistema de posicionamento universal e confiável. Porém, erros inerentes a este sistema contribuem para a falta de acurácia (apenas 100m). Estações de base diferencial podem melhorar a acurácia para valores de até menos de 2m; entretanto podem gastar muito tempo para analisar e ficam restritos a uma determinada área.
- "Bluetooth"

  É um padrão industrial para comunicações entre dispositivos sem fio como celulares, PDA's, computadores portáteis e também entre um computador com capacidade para o "bluetooth" e seus periféricos. Um único dispositivo "Bluetooth" pode ser capaz de realizar ligações telefônicas, sincronizar informações com computadores, receber e enviar FAX ou mesmo imprimir documentos.

O padrão "Bluetooth" opera na freqüência de 2.45GHz e seu sinal tem alcance de cerca de 10m.

- Localização via celular

  É uma tecnologia já utilizada em países como Japão, Coréia, China e Estados Unidos e que agora chegou ao Brasil. A posição do telefone celular é calculada através da tecnologia gpsOne, que combina informações obtidas pelos satélites GPS com informações do celular medidas pela rede CDMA.

  A precisão obtida nas localizações depende do ambiente no qual o aparelho se encontra no momento em que é localizado, podendo apresentar um erro de 10m a 30m quando em ambientes abertos ou erros maiores quando em ambientes fechados.

  Outro método de localização por celular em redes GSM é realizado pela identificação da célula onde o cliente está. Neste caso a imprecisão é de 100m.

- Wireless 802.11

  Termo utilizado para descrever telecomunicações nas quais as ondas infravermelhas, de rádio ou eletromagnéticas carregam sinais a serem transmitidos em um meio sem fio. É um tipo de tecnologia para transmissão de dados. Sob o padrão 802.11b, na prática ele possui uma abrangência de 20m em ambientes internos e 50m nos externos. A localização com esta tecnologia geralmente utiliza a triangulação e leva em conta os percursos possíveis do sinal, o espalhamento e a atenuação dos sinais. Mas como isso tem muito custo computacional, algumas soluções comerciais utilizam um banco de dados para armazenar as intensidades de amostras de vários pontos no mapa; cada ponto de amostra contém estatísticas das intensidades dos sinais recebidos, melhorando desta forma o desempenho, de modo que pode conseguir uma acurácia da ordem de 1m, sob certas condições.

Neste projeto, a função de localização do usuário, dada pelos dispositivos móveis, será realizada através de uma rede Wireless utilizando o protocolo 802.11 (padrão IEEE) (HAEBERLEN et al., 2004). Este sistema também é conhecido como LBS (Location-based Services). (LADD et al., 2004).

Cada dispositivo móvel faz a comunicação com o servidor de localização e este determina a posição atual do dispositivo. O servidor é capaz de informar as coordenadas x, y, z e também a velocidade e a direção do movimento. (ELNAHRAWY; LI; MARTIN, 2004) e (XIANG et al., 2004). Neste trabalho, usou-se somente a informação (x,y) de posição e z para definir o andar.

Este sistema de localização necessita de no mínimo 3 pontos de acesso Wi-Fi e dispositivos clientes com placas Wi-Fi.

Atualmente existem tanto softwares comerciais (Ekahau, BlueSoft) como abertos (PlaceLab) que realizam as tarefas descritas acima.

Pesquisas sobre a acurácia na localização via Wi-Fi mostraram que o PlaceLab (PLACELAB, 2005) era menos preciso que ferramentas comerciais. Enquanto o PlaceLab oferece de 10 a 20m de erro na localização, ferramentas como o Ekahau garantem 1m de precisão.

Assim sendo, como uma aplicação em ambientes internos necessita de grande precisão, optou-se por utilizar o Ekahau neste projeto.

No entanto, ressalta-se que o sistema é flexível para que seja possível uma adaptação a outros sistemas de localização, já que os algoritmos de localização via Wireless 802.11 ainda estão em fase de pesquisa e desenvolvimento.

2.5 Interface com o Usuário

A interface com o usuário é gráfica e apresentada através de applets em Java para que não seja necessária a instalação do sistema cliente pelo usuário.

A interface apresenta para o usuário um conjunto de locais possíveis de serem visitados. Feita a seleção do local destino, é mostrado um mapa contendo o ponto onde o usuário se encontra e a rota determinada pelo sistema até o ponto de destino desejado.

O usuário terá acesso ao sistema através de uma interface web, e deverá acessar seu site para ter acesso ao serviço. Um applet será carregado permitindo assim ao usuário a visualização do resultado fornecido pelo sistema.

Como primeira entrada, o sistema irá solicitar que o usuário informe se ele possui alguma dificuldade de locomoção, como o uso de cadeira de rodas, muletas, carga de objetos de dimensões extensas, etc. Esta informação será utilizada pelo algoritmo de roteamento para o cálculo do custo de cada um dos caminhos possíveis.

Em seguida, o sistema exibe a lista de opções de locais de destino e seus respectivos sub-itens dentro do estabelecimento considerado. Para o caso do Prédio da Engenharia Elétrica da Poli, a lista apresentada tem as seguintes opções:

1o Menu	2o Menu	3o Menu
Salas de Aula	Bloco A	A1-01
		...
	Bloco B	B2-01
		...
	Bloco C	C1-01
		...
	Bloco D	D1-01
		...
Salas de Professor	Anna Reali	
	Jorge Risco	
	Selma Melnikoff	
	...	
Laboratórios	PCS-LARC	
	PCS-LTS	
	Lab. Proj. Formatura	
	...	
Secretarias	PCS	
	PSI	
	...	
Sanitários	Masculino	
	Feminino	
Biblioteca		
Anfiteatro		
Pró-aluno		
Lanchonete		

Definido o local de destino, o usuário deve pressionar um botão de confirmação e o sistema calcula a melhor rota a ser percorrida. É exibido um mapa com a rota no menor nível hierárquico possível de ser representado para o par (local de origem, local de destino) determinado. Para o caso de caminhos que envolvam nós de diferentes níveis hierárquicos, é possível clicar nos nós que contêm um sub-mapa e exibir o mesmo.

O mapa exibido pode ser aumentado (Zoom In e Out) para tornar-se visível no caso de mapas extensos, e navegável devido às dimensões restritas da tela do PDA. Um exemplo de tela com a rota recomendada pode ser visto na .

Figura 2.4 – Exemplo de tela apresentada ao usuário.

2.6 *Hardware e Software necessários*

Será apresentada a seguir a relação dos hardwares e softwares utilizados no desenvolvimento do projeto.

2.6.1 Hardware

Foram utilizados os seguintes componentes:

- 1 PDA ou notebook: Este dispositivo tem a função de interface entre o sistema e o usuário. A comunicação entre eles é feita via rede wireless, a qual também possibilita que o sistema localize o usuário no estabelecimento.

  PDA:

  - Modelo: HP Pocket PC iPAQ h5550
  - Especificação:
    - Sistema Operacional: Windows Mobile
    - Processador: Intel 400MHz (com tecnologia Xscale™)
    - Conectividade: WLAN 802.11b, Bluetooth®
    - Memória: 128 SDRAM, 48 Flash ROM
    - Display: 3,8", 65k cores, resolução 240x320

  Notebook:

  - Especificação:
    - Sistema Operacional: Windows XP
    - Processador: Intel Pentium III ou IV
    - Conectividade: WLAN 802.11b
    - Memória mínima: 256 MB RAM
- 3 Access Points: Estes equipamentos são utilizados na infra-estrutura do sistema, provendo a rede wireless para a comunicação entre o usuário e o servidor central e auxiliando no mecanismo de localização do usuário. Foi utilizada a infra-estrutura já existente no estabelecimento devido à existência da quantidade mínima de três access points para possibilitar a localização do usuário.
- Servidor central (PC): Responsável pelo processamento dos algoritmos de roteamento e pelo controle do sistema de localização.

2.6.2 Software

Foram usadas as seguintes ferramentas:

- IDE Eclipse: Plataforma aberta para desenvolvimento de software que foi utilizada para auxiliar todo o processo de implementação do sistema.
- J2SDK: Compilador Java.
- CVS: foi utilizado para fazer o controle de versões do código fonte e documentos em geral. Neste projeto foi utilizado um plano gratuito do CVS Dude, que é um servidor de CVS (Concurrent Versions System).
- IBM Rational Rose: Este software foi utilizado para elaborar partes da especificação de requisitos de software, tais como diagramas de casos de uso, classes e seqüência.
- AutoCAD: Este software foi utilizado para a manipulação dos arquivos das plantas do Prédio da Elétrica.
- Microsoft Project 2000: Este software não foi utilizado no desenvolvimento do projeto propriamente dito, e sim para a organização do andamento do projeto. Nele é possível enumerar as tarefas envolvidas em cada fase do projeto, delegar estas tarefas a integrantes da equipe e elaborar cronogramas detalhados do projeto, entre outras funções.

3 DETALHAMENTO E IMPLEMENTAÇÃO

As tecnologias selecionadas na fase de especificação foram estudadas mais profundamente e aplicadas ao desenvolvimento dos módulos do sistema. O modelo de classes do sistema pode ser observado no Anexo B.

A partir dos modelos simplificados criados na fase anterior, foi possível implementar melhorias e novas funcionalidades necessárias ao sistema, como o desenvolvimento do algoritmo de roteamento completo a partir da versão inicial de testes, e a geração do applet final a partir do protótipo da interface.

3.1 Representação do Mapa

Conforme dito no capítulo 2, a forma mais conveniente de representação de mapa para este projeto foi a representação topológica através de grafos. Sua escolha pode ser justificada pela adequação à modelagem abstrata do ambiente e pela ausência de dados métricos para a execução do algoritmo de roteamento, tratado mais adiante.

3.1.1 Manipulação das Plantas

Para a criação do grafo, foi necessário obter as plantas do Prédio da Elétrica. Mesmo que de forma simplificada, tais plantas deveriam indicar não apenas os pontos de destino do usuário, mas principalmente todos os caminhos de circulação dentro do estabelecimento, permitindo assim a obtenção dos dados fundamentais ao algoritmo de roteamento.

As plantas obtidas representavam o Prédio da Elétrica dividido em cada um de seus blocos, e estes divididos em seus respectivos andares. Como esses arquivos tinham sido criados pela CIPA (Comissão Interna de Prevenção de Acidentes), as imagens possuíam algumas informações que não seriam utilizadas neste projeto, as quais foram posteriormente eliminadas para melhor clareza das plantas.

Apesar de as imagens atenderem aos requisitos de representatividade mencionados anteriormente, os arquivos possuíam alguns dados desatualizados devido à data de criação dessas plantas. Assim, foi preciso fazer uma revisão das plantas, comparando os dados representados com as reais instalações do Prédio da Elétrica. As diferenças observadas foram corrigidas nos arquivos através do software AutoCAD.

Finalizada esta etapa de atualização, as imagens das plantas tiveram que ser convertidas para um formato compatível com o software de localização, cujo funcionamento será detalhado mais adiante. Dessa forma, foi escolhido o formato GIF para as imagens, devido à sua capacidade de compressão, produzindo arquivos finais cujos tamanhos não exigissem alto processamento e memória dos computadores utilizados. Os mapas usados encontram-se no Anexo C.

Em seguida, as imagens pertencentes a um mesmo andar e que possuíssem interligação real no Prédio da Elétrica foram unidas em um mesmo arquivo, de forma a permitir que o algoritmo de roteamento interpretasse as plantas corretamente.

3.1.2 Armazenamento dos Dados

Uma outra etapa relacionada à representação de mapa consiste em definir a forma como os dados do grafo produzido são apresentados ao algoritmo de roteamento. As opções existentes eram a criação de um banco de dados ou a geração de um arquivo XML, ambos com o objetivo de armazenar os dados de cada um dos nós, a interligação entre eles, entre outras informações.

A solução adotada no projeto foi o armazenamento do grafo através de um arquivo XML, pois este permitiria uma maior facilidade de acesso às informações do grafo, além de eliminar a necessidade da utilização de um aplicativo de banco de dados no servidor que comportasse o módulo de roteamento.

Para suportar esta decisão de projeto, foram desenvolvidas aplicações para geração e interpretação do arquivo XML. A primeira possuía uma interface gráfica que permitia que o usuário indicasse, através do clique do mouse, o ponto da imagem que representaria um nó do grafo, podendo utilizar algumas funcionalidades como

zoom para melhorar a precisão na definição do nó, além de indicar as informações relativas ao mesmo através dos campos de entradas de dados fornecidos pela aplicação. Após a determinação dos nós pode-se definir a integração entre eles, gerando assim as informações relativas aos arcos do grafo.

A aplicação criada para interpretar o arquivo XML extrai deste as informações dos nós do grafo hierárquico, sendo que a leitura deste XML é realizada uma única vez na inicialização do sistema, armazenando toda a representação do grafo em memória para agilizar as futuras consultas durante o processo de roteamento, eliminando a necessidade de se fazer acesso ao arquivo freqüentemente.

Quando o usuário faz uma solicitação ao sistema, o módulo de roteamento obtém as informações relativas às restrições e ao local de destino escolhido, e juntamente à representação armazenada, realiza os cálculos para encontrar a melhor rota que atenda às necessidades deste.

Ambas as aplicações para geração e interpretação do arquivo XML foram criadas utilizando a linguagem JAVA. As principais características da linguagem que definiram sua escolha para o desenvolvimento das aplicações foram seus recursos gráficos, usados na interface da aplicação.

A etapa final da representação de mapas consiste em, após a obtenção das plantas atualizadas e dos aplicativos que permitem seu uso, gerar o arquivo XML final do Prédio da Elétrica. Tal processo, apesar de sua simplicidade, mostrou-se bastante demorado devido ao grande número de nós que o estabelecimento possui.

3.1.3 Criação dos Grafos

Para que o algoritmo de roteamento criado funcione corretamente, ele se utiliza de um grafo hierárquico do ambiente a ser explorado. Dessa forma, foi necessário definir a hierarquia que representaria o Prédio da Elétrica, ou seja, definir quantos seriam os níveis hierárquicos, o que cada um deles representaria e como seria feita a divisão dos nós.

A idéia inicial de hierarquia foi a seguinte:

- Nível 0: a POLI como sendo o nível hierárquico mais alto;
- Nível 1: o Prédio da Elétrica (e os outros prédios) como sendo o primeiro nível intermediário;
- Nível 2: cada um dos blocos como sendo o segundo nível intermediário;
- Nível 3: cada andar dentro de um bloco como sendo o nível hierárquico mais baixo.

Entretanto, percebeu-se que tal divisão não seria adequada para a estrutura apresentada pelo Prédio da Elétrica, pois seus blocos possuem interconexões não somente no andar térreo, mas também entre outros andares dos diferentes blocos. Assim, foi necessário modificar a hierarquia acima apresentada de forma que um bloco como um todo não fosse considerado.

A hierarquia final utilizada para representar o Prédio da Elétrica está ilustrada na Figura 3.1 e possui:

- Nível 0: a POLI como sendo o nível hierárquico mais alto;
- Nível 1: o Prédio da Elétrica (e os outros prédios) como sendo o primeiro nível intermediário;
- Nível 2: cada andar de um dado bloco como sendo o nível hierárquico mais baixo.

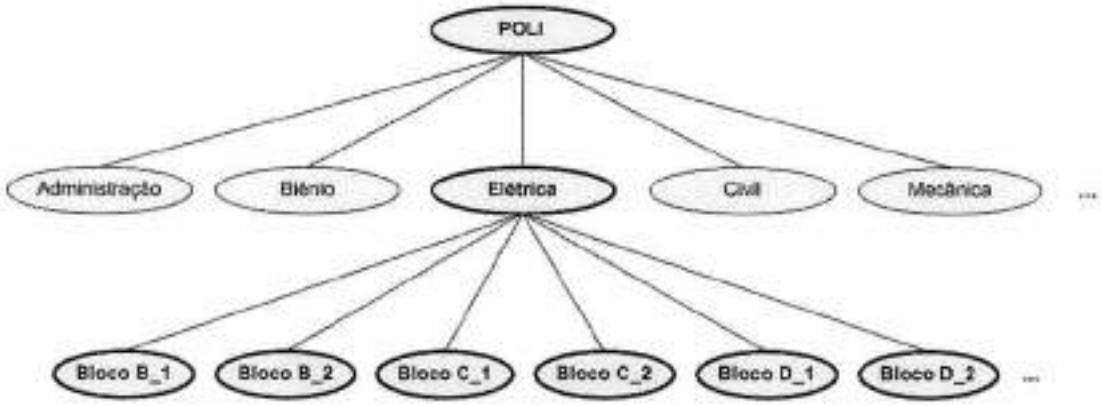


Figura 3.1 – Hierarquia de localidades.

Os grafos utilizados para a implementação do projeto podem ser observados no Anexo D.

3.2 Algoritmo de Roteamento

No início da fase de implementação, após o estudo de diversos tipos de algoritmos de roteamento e a escolha da solução mais adequada para o caso em questão, foi desenvolvido um algoritmo muito simples com a finalidade de permitir o desenvolvimento já integrado dos diversos módulos do sistema.

Após essa implementação inicial partiu-se para o desenvolvimento do algoritmo propriamente dito, que havia sido escolhido anteriormente, baseado num algoritmo utilizado em cadeiras de rodas inteligentes (CAGIGAS; ABASCAL, 2004), sendo que foi implementado utilizando a linguagem de programação Java, conforme planejado.

Sobre a idéia original do algoritmo utilizado na cadeira de rodas inteligente foram adicionadas algumas características para torná-lo mais genérico e adequado às necessidades do projeto, tais como a utilização de custos variáveis para os arcos do grafo e de arcos direcionais, fazendo todas as adaptações necessárias no método de busca.

A importância em se utilizar custos variáveis é a de possibilitar o tratamento de diferentes perfis de usuários, adequando os caminhos informados às necessidades dos mesmos, personalizando a solução caso a caso. Para implementar esta funcionalidade utilizou-se um vetor de custos para cada arco, conseguindo assim mapear a quantidade de restrições ou perfis necessários. Para cada arco, há um custo específico para cada perfil considerado no sistema, fazendo com que o caminho obtido seja o de menor custo para cada um dos perfis.

Nesta nova representação do arco, este é descrito pelo vértice origem, do destino, e possui um vetor de custos associado, sendo que o tamanho deste vetor deve ser igual ao número de perfis desejados, e cada posição do vetor deve conter o custo relativo a um dos perfis. Sendo assim, pode-se mapear um arco que representa uma escada com um custo baixo para o perfil "sem restrições" e um custo

relativamente alto para o perfil "cadeirante", de forma a forçar o sistema a apresentar um caminho com escada para este último perfil somente se não houver outra alternativa mais viável e adequada às suas necessidades.

Na inicialização do sistema, o grafo já armazenado em memória contém a informação dos custos para cada perfil. Ao realizar-se uma consulta, o perfil é passado como parâmetro, sendo que esse é informado pelo usuário no início do acesso ao sistema e mantido durante a utilização do mesmo, sendo o applet responsável por transmiti-lo ao servidor no momento da solicitação de uma rota.

Já as modificações realizadas para transformar os arcos não-direcionais em direcionais são importantes para os caminhos de "mão única", como no caso de uma escada rolante. O cuidado especial que deve ser observado nesta modificação é que, neste caso, os caminhos não podem ser construídos de trás para frente, ou seja, não poderíamos construir o caminho a partir do destino, pois a volta não é válida.

O algoritmo trabalha com a construção de um esqueleto de caminho sobre o grafo hierárquico, e somente após esta etapa o caminho é completado. Este esqueleto de caminho é construído baseando-se em um conjunto de vértices especiais (os chamados nós pontes), sendo que na determinação deste esqueleto o fato de termos arcos direcionais não atrapalha, pois na etapa em que o caminho é completado, o mesmo é construído no sentido correto, começando da origem e seguindo até o destino.

Na inicialização do sistema é realizada a leitura do arquivo XML que contém as informações referentes ao mapa do local, ou seja, o grafo hierárquico com todos os níveis da hierarquia proposta, todos os vértices e todos os arcos com os respectivos custos. Depois de realizada esta leitura e inicialização das variáveis correspondentes são pré-calculados alguns caminhos "especiais", para permitir um melhor desempenho das buscas.

São três os casos em que devem ser pré-calculados os caminhos:

1. Caminhos que conectam dois nós pontes dentro de um submapa;
2. Caminhos que conectam nós pontes de um submapa com os nós pontes de seu submapa pai; e

3. Caminhos que conectam submapas irmãos contidos num submapa. Dois submapas são irmãos se ambos possuem o mesmo submapa pai.

Os caminhos pré-calculados evitam o recálculo destes sub-caminhos no processo de busca hierárquico. Por um lado é necessário mais espaço em memória para armazenar estes caminhos, mas por outro, ele pode garantir melhor desempenho e caminhos otimizados.

O procedimento utilizado para pré-calcular os caminhos pode ser observado no Anexo E, assim como todo o algoritmo de roteamento.

3.2.1 Replanejamento da rota

Durante o desenvolvimento do sistema, foi observada a importância do replanejamento da rota, pois se trata de uma parte complementar ao algoritmo de roteamento que pode requerer muitos recursos computacionais quando a quantidade dos nós de interesse aumenta.

A fim de diminuir a troca de informação entre o cliente e o servidor, foi escolhido trabalhar a lógica de replanejamento no primeiro, pois, uma vez que a solicitação do caminho é feita no cliente, é interessante que ele mesmo seja capaz de detectar se o usuário está seguindo o caminho correto, eliminando assim o excesso de mensagens para avaliar o comportamento do usuário em relação à rota obtida.

Para identificar se o cliente desviou-se da rota informada pelo servidor, foram considerados dois modelos de avaliação. O primeiro consiste em estipular dois caminhos delimitadores, paralelos ao caminho sugerido e espaçados de uma determinada distância, determinando logo no início toda a área permitida ao usuário, e solicitando o replanejamento se o mesmo saísse desta área. Como este modelo possui um elevado custo computacional, devido à necessidade de cálculos que envolvem a transformação das rotas obtidas em funções matemáticas, a geração da área permitida completa baseada nesta função e a avaliação da posição do usuário em relação a todas estas funções matemáticas, optou-se por utilizar um modelo de avaliação mais simplificado e eficiente.

O modelo adotado então consiste em estipular uma circunferência centralizada no ponto do caminho mais próximo da posição do usuário, e o raio é obtido a partir da distância entre este ponto e o próximo multiplicada por um coeficiente que determina a tolerância permitida. A avaliação é realizada então com relação a esta circunferência, com uma freqüência pré-definida, sendo que quando a função de avaliação identifica que o usuário saiu da circunferência definida, calcula-se uma nova circunferência com relação ao próximo ponto do caminho. Se o usuário estiver dentro da nova circunferência, é considerado que o usuário está no caminho. Caso contrário uma nova solicitação do caminho é feita para replanejamento do mesmo. Esse processo é repetido até que o usuário atinja o destino.

Este modelo de avaliação possui uma implementação mais fácil, uma vez que o mesmo necessita somente do cálculo da distância entre a posição atual e o ponto de avaliação e da comparação numérica entre esta distância e o raio permitido.

Na Figura 3.2 é possível ter uma idéia mais clara da simplicidade e eficiência do algoritmo utilizado. Nesta observa-se o caminho que liga o vértice 1 ao 3. Inicialmente, quando o usuário está próximo ao vértice 1, define-se uma circunferência ao redor do mesmo onde ele pode estar localizado (Área A1), sendo que quando o usuário sai desta área, verifica-se se ele está na área A2. Se ele estiver, então ele está seguindo a rota corretamente e nada precisa ser feito, caso contrário é solicitada nova rota. Da mesma maneira quando o usuário deixar a área A2, será delimitada uma nova área ao redor do vértice 3, e verificado se o usuário está seguindo a rota. O raio utilizado pode ser ajustado através de um coeficiente, possibilitando a determinação da tolerância permitida.

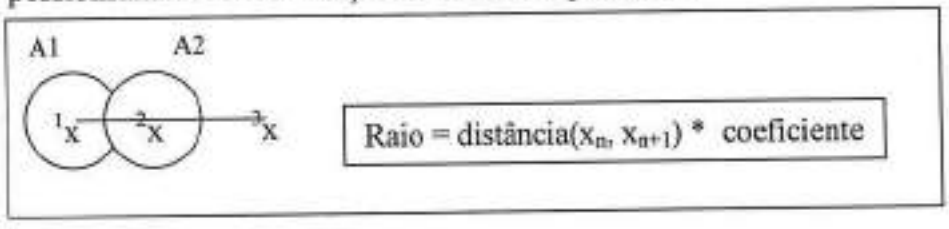


Figura 3.2 – Replanejamento.

3.3 WLAN e Sistema de Localização

A função principal deste módulo é de oferecer a localização do usuário através de algumas tecnologias disponíveis no mercado. Pensou-se em várias possibilidades, como por exemplo: rede 802.11, rede Bluetooth, sistema GPS, análise de imagens de vídeo, malha de sensores de localização entre outros.

Comparando essas tecnologias, a malha de sensores pode oferecer uma precisão alta na localização, mas tem um custo muito alto de implantação. Já a análise de imagens de vídeo apresenta-se como uma tecnologia mais moderna, podendo possuir algum aplicativo disponível, entretanto seriam necessários equipamentos mais robustos para realizar esta função. O sistema GPS tem muita maturidade no mercado e vários exemplos de uso, porém é feito para ser utilizado em locais abertos, não funcionando no interior de estabelecimentos fechados devido à falta de visibilidade dos satélites. Quanto à rede Bluetooth, esta possui pequeno alcance e existem alguns projetos de localização baseados nesta tecnologia, mas o problema de utilizá-la é que uma malha condensada de pontos de acesso desta tecnologia pode levar a um custo alto de implantação.

Já a rede 802.11, tecnologia que permite a conexão entre computadores e redes através da transmissão e recepção de sinais de rádio, oferece uma rede de maior abrangência e uma alta velocidade de transmissão de dados. Algumas tecnologias de localização utilizam sinais desta rede e têm uma precisão suficiente para viabilizar alguns projetos com o uso da mesma. Algumas técnicas são aplicadas para isso, como triangulação ou propagação dos sinais, mas estas podem sofrer com interferência, reflexão, barreiras e espalhamento das ondas de rádio.

O pacote Ekahau é um produto comercial que realiza esta função de localização através de um sistema auxiliar (Ekahau Positioning Engine) de calibração para coletar pontos de amostra da rede sem fio em vários lugares. Algumas informações relevantes são armazenadas no modelo de posicionamento como, por exemplo, estatística da intensidade do sinal no lugar de amostragem e coordenadas relacionais no mapa, para permitir um rastreamento do usuário com precisão.

O Ekahau contém quatro módulos: o Gerenciador Ekahau (*Ekahau Manager*), o Servidor Ekahau (*Ekahau Server / Ekahau Positioning Engine*), o SDK/YAX Ekahau

e o Cliente Ekahau (*Ekahau Client*). A comunicação entre esses módulos pode ser observada na Figura 3.3.

O Gerenciador Ekahau é uma aplicação para criar modelos de posicionamento, salvando-os no Servidor Ekahau; delimitar áreas lógicas no mapa; testar posicionamento em tempo real; e analisar a precisão.

O Servidor Ekahau oferece informações de posição de equipamentos e dispositivos que possuem suporte a wireless, de áreas lógicas onde estes se encontram e de velocidade e direção destes, através de uma interface Java (Ekahau SDK), do protocolo YAX ou do Gerenciador Ekahau.

O SDK/YAX Ekahau é um módulo para desenvolvimento, que contém o SDK (*Software Development Kit*), as documentações Javadoc, e alguns exemplos de uso do mesmo. Para aplicações que não são desenvolvidas em Java, o protocolo YAX é uma interface alternativa que utiliza diretamente o *socket* do TCP.

O Cliente Ekahau é instalado nos dispositivos móveis para que o Gerenciador possa localizá-los via rede 802.11. Ele instala também um protocolo Ekahau da camada 3 do modelo OSI de rede para realizar o rastreamento do cliente na calibração e na localização.

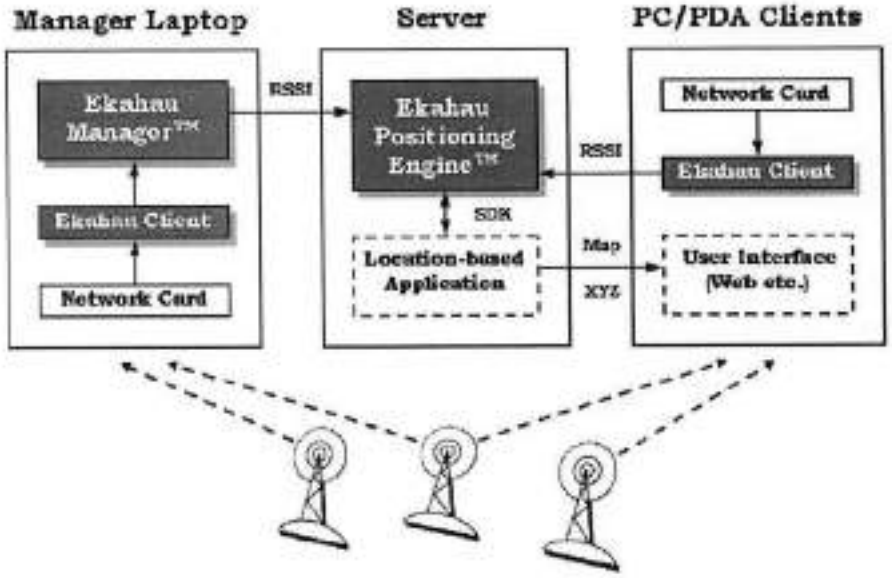


Figura 3.3 – Módulos do Ekahau (retirada de EKAHAU (2005)).

Para utilizar o pacote Ekahau no projeto, desenvolveu-se no módulo de localização uma interface entre o sistema principal e o Ekahau, para que o sistema seja isolado dos detalhes do uso do Ekahau, garantindo assim a transparência deste módulo e a possibilidade de substituição do mesmo futuramente.

Como o Gerenciador Ekahau é responsável pela localização, para se obter a posição de um determinado cliente é criada uma instância para esperar o evento de rastreamento, cujo intervalo depende do *driver* do cliente. Através do evento capturado são atualizadas as informações do dispositivo rastreado.

Para que o cliente possa adquirir a informação de localização independentemente do intervalo de rastreamento pelo Cliente Ekahau, é criada uma tabela de localização com as informações relevantes de posição dos dispositivos que estão utilizando o sistema, sendo que as informações antigas são descartadas da tabela para economizar a memória.

3.4 Interface com o Usuário

A interface com o usuário proposta é através de uma página *web* que contém menus subdivididos por tipos de salas (salas de aula, anfiteatros, salas de professores, sanitários, etc), sendo que o usuário pode navegar através destes para encontrar a localização de determinado local. Através desse mesmo site, o usuário pode então solicitar que lhe seja informado um caminho para chegar àquele local. Quando este caminho é solicitado, um *applet* Java é carregado para possibilitar a integração desta interface que informa o caminho ao usuário com a página *web*.

No desenvolvimento do site foi utilizada a tecnologia ASP.NET, sendo que esta interface obtém as informações do estabelecimento através da base de dados implementada em MySQL para a montagem dos menus e exibição de todas as informações do site.

Na tela que especifica o destino escolhido pelo usuário, existe um botão para chamar a página que exibe o caminho, sendo passado como parâmetro neste caso o ponto que o usuário almeja e o seu perfil (suas restrições).

Este *applet* começou a ser implementado ainda na fase de análise, pois foi utilizado como protótipo, para demonstrar a interface gráfica que seria utilizada no projeto. Inicialmente havia sido implementado um *applet* simples que carregava uma imagem e desenhava um caminho (previamente informado) em cima desta imagem, ainda sem comunicação com o servidor e nenhuma lógica de negócio.

Após essa implementação inicial foi acrescentada a este *applet* a comunicação cliente-servidor, através do RMI, sendo possível assim solicitar a localização e a rota diretamente para o servidor, sendo que este último inicialmente respondia com uma localização "forjada" e o algoritmo de roteamento ainda não era o final.

Com isso pode-se começar a desenvolver a lógica de negócio do *applet*, que consiste em obter a localização do servidor, solicitar o caminho a ele e ficar obtendo a localização de tempo em tempo para avaliação do desvio do usuário em relação à rota apresentada e possível solicitação de recálculo do caminho se o usuário desviar do caminho apresentado inicialmente. Este desvio é calculado sempre em relação à próxima aresta do grafo da rota que o usuário deveria percorrer, gerando uma área onde o usuário pode estar e, quando ele sai desta área, verifica-se se ele está na próxima aresta (ou seja, no caminho correto) ou se é necessário recalcular a rota.

3.5 Arquitetura Cliente-Servidor

Quando foram estudadas as diversas soluções para a comunicação entre o cliente (applet) e o servidor (de roteamento e localização), pensou-se em três possibilidades: implementar um protocolo de comunicação, desenvolver web services ou utilizar RMI (Remote Method Invocation) (SUN MICROSYSTEMS, 2005).

Estudando mais profundamente essas soluções foram sendo detectados pontos positivos e negativos das mesmas, e concluiu-se que no primeiro caso, o trabalho para desenvolver todo o protocolo de comunicação para permitir as trocas de

informação, tanto de rotas como de localização do dispositivo, seria muito dispendioso e não traria grandes vantagens. A única vantagem nesse caso seria uma pequena economia de banda, mas considerando que o sistema estará rodando numa LAN (Local Area Network) esse tráfego a menos não é significativo e, portanto, não justifica o esforço necessário.

Desenvolvendo web services para prover essa infra-estrutura cliente-servidor seria interessante para possibilitar a integração futura com módulos que utilizem outras tecnologias que não o Java, porém para o escopo do projeto, com o lado cliente necessitando das rotas e localização, seria desnecessário haver tanta sofisticação como web services, sendo que novamente achou-se que não compensaria seguir essa linha.

Com isso, após essa fase de estudo sobre essas três tecnologias, optou-se por desenvolver a arquitetura cliente-servidor baseada no uso do RMI, que é uma especificação do Java para realizar chamadas de métodos remotamente. O RMI possibilita ao programador o desenvolvimento de aplicações distribuídas baseadas na tecnologia Java, onde métodos de objetos Java remotos podem ser invocados de outras máquinas virtuais Java, possivelmente em diferentes domínios de rede. O RMI utiliza serialização dos objetos para encapsular e desencapsular parâmetros, suportando verdadeiramente polimorfismo orientado a objeto.

3.5.1 A tecnologia RMI

Sistemas distribuídos demandam que aplicações computacionais rodando em diferentes espaços de endereçamento (redes distintas) estejam aptas a se comunicar. Para um mecanismo de comunicação básico a linguagem de programação Java suporta *sockets*, que é flexível e suficiente para comunicação de modo geral. Todavia, *sockets* necessita que tanto cliente como servidor implementem protocolos no nível de aplicação para codificar e decodificar mensagens que possibilitem a comunicação, e o projeto destes protocolos é custoso e é mais uma fonte de erros para a aplicação.

Uma alternativa ao *socket* é a chamada de procedimentos remotos (RPC - Remote Procedure Call), que abstrai a interface de comunicação para o nível de uma chamada de procedimento. Ao invés de trabalhar diretamente com *sockets* o programador implementa como se estivesse chamando um procedimento local, quando de fato os argumentos da chamada estão sendo encapsulados e enviados para o destino remoto da chamada. Sistemas RPC codificam argumentos e valores de retorno usando uma representação de dados externa (St LAURENT; JOHNSTON; DUMBILL, 2001).

RPC, no entanto, não é bom em sistemas com objetos distribuídos, onde a comunicação no nível de aplicação de objetos residentes em diferentes espaços de endereçamento se faz necessária. Para suprir estas necessidades, sistemas de objetos distribuídos necessitam de chamada de métodos remotos ou RMI. Nesses sistemas, um objeto local chamado de *stub* gerencia a chamada num objeto remoto. Na Figura 3.4, encontra-se a arquitetura do RMI confrontada com o modelo OSI.

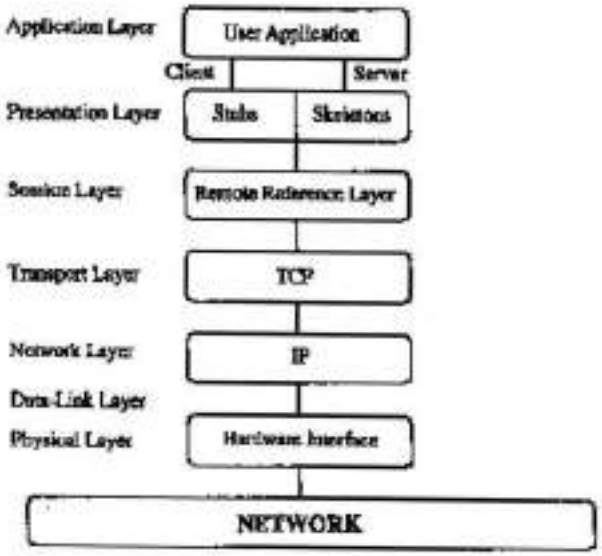


Figura 3.4 – RMI no modelo de referência OSI (retirada de MAHMOUD (2000))

As metas para o suporte a objetos distribuídos na linguagem de programação Java são as seguintes (SUN MICROSYSTEMS, 2005):

- Suportar consistentemente a invocação remota de objetos em diferentes máquinas virtuais.
- Suportar chamadas de retorno de servidores para applets.
- Integrar o modelo de objetos distribuídos na linguagem de programação Java de uma maneira natural mantendo a maior parte da semântica dos objetos desta linguagem.
- Deixar aparente as diferenças entre o modelo de objetos distribuídos e o modelo de objetos locais da plataforma Java.
- Tornar a escrita de aplicações distribuídas o mais simples possível.
- Preservar o chamado "*type-safety*" (não permitir que variáveis de um tipo sejam associadas com variáveis de outro tipo qualquer inconsistentemente) que o ambiente de execução da plataforma Java provê.
- Suportar várias semânticas de referência para objetos remotos; por exemplo, referências não persistentes, referências persistentes, e *lazy activation*.
- Manter o ambiente seguro que a plataforma Java provê através do gerenciamento da segurança (security managers) e dos class loaders.
- E, fundamentando todas essas metas, há um requisito geral que o modelo RMI deve ser simples (fácil de usar) e natural (se ajustar bem na linguagem).

O RMI utiliza um mecanismo padrão (empregado em sistemas RPC) para comunicação com objetos remotos: *stubs* e *skeletons*. Um *stub* atua como um representante local para o objeto remoto. O cliente invoca um método no stub local cuja responsabilidade é carregar esta chamada de método para o objeto remoto. No RMI, um *stub* implementa o mesmo conjunto de interfaces que o objeto remoto.

Quando um *stub* é invocado ele faz o seguinte:

- Inicia a comunicação com a JVM remota que contém o objeto remoto;
- Encapsula (escreve e transmite) os parâmetros para a JVM remota;
- Espera pelo resultado do método invocado;
- Desencapsula (lê) o valor de retorno ou exceção lançada;
- Retorna o valor ao cliente.

O stub esconde a serialização de parâmetros e a comunicação no nível de rede para fornecer um mecanismo simples de invocação de método remoto para o cliente.

Na JVM remota, cada objeto remoto pode ter um skeleton correspondente. O skeleton é responsável por encaminhar a chamada para a implementação do objeto remoto atual. Quando um skeleton recebe uma invocação de método ele faz o seguinte:

- Desencapsula (lê) os parâmetros para o método remoto;
- Invoca o método na implementação do objeto remoto atual;
- Encapsula (escreve e transmite) o resultado (valor de retorno ou exceção), enviando-o para o cliente.

No Java 2 SDK, Standard Edition, v1.2 um protocolo adicional foi incluído para o *stub*, eliminando a necessidade de *skeletons* em ambientes puramente Java 2 platform. Ao invés disso um código genérico é responsável por realizar as atividades do *skeleton* do JDK1.1. *Stubs* e *skeletons* são gerados pelo compilador rmic.

Esta interação entre *stubs* e *skeletons* é apresentada na Figura 3.5.

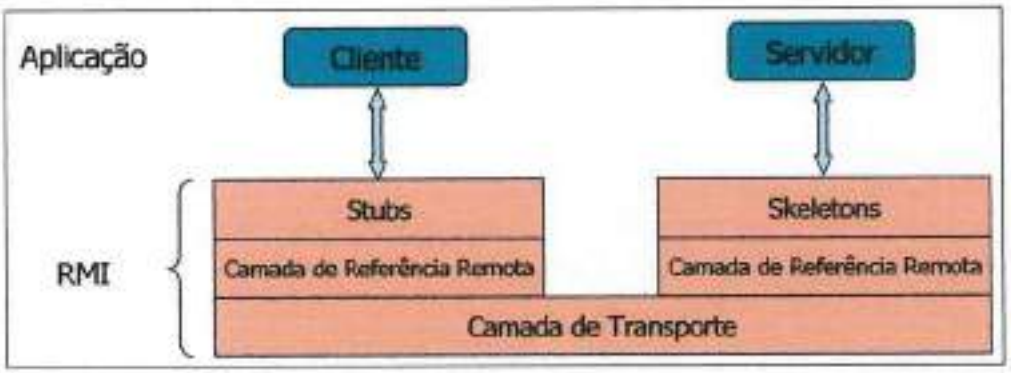


Figura 3.5 – Arquitetura RMI
(baseada em SUN MICROSYSTEMS (2005)).

3.6 Arquitetura final do sistema

A arquitetura do sistema é apresentada na e mostra toda a organização e comunicação do sistema. Pode ser observada desde a parte da comunicação do navegador (browser) do cliente com o servidor WEB, quando este inicia o acesso ao sistema e navega pelas páginas do site, até a utilização dos sistemas de localização e roteamento propriamente ditos pelo Applet que é carregado no computador do usuário quando este solicita a informação de como chegar a determinado local.

Toda a comunicação do Applet com o servidor é feita através de RMI (Remote Method Invocation), e tanto o site como o servidor principal estão acessíveis via rede wireless 802.11, para permitir mobilidade ao usuário.

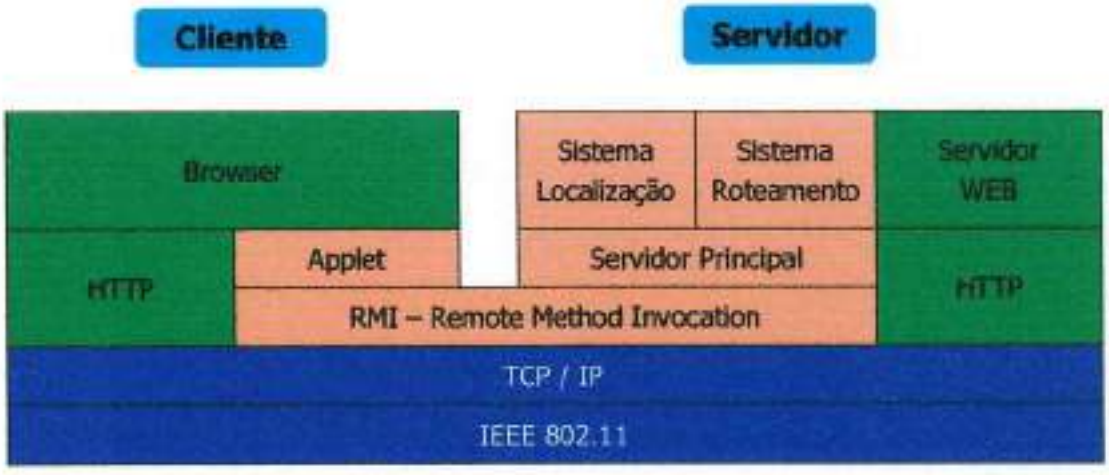


Figura 3.6 – Arquitetura do Sistema.

4 RESULTADOS

Neste capítulo serão apresentados os testes realizados com a análise dos resultados obtidos, apresentando os problemas ocorridos nos mesmos. Primeiramente serão mostrados os testes realizados nos módulos individualmente, após a implementação dos mesmos, e em seguida serão apresentados os testes dos módulos do sistema integrados.

4.1 Testes do interpretador de XML

Após a implementação do interpretador de XML, que é o responsável pelo carregamento das informações de todo o grafo hierárquico, foi realizado um teste de conformidade do mesmo para verificar a interpretação correta do arquivo. Para isso, com o auxílio de um XML com um grafo hierárquico qualquer, foi possível verificar que o interpretador estava carregando todos os vértices e arcos corretamente e que o tempo gasto era compatível com as necessidades do projeto.

Com estes resultados satisfatórios, esse módulo já passou a ser utilizado pelas outras partes do sistema, carregando o grafo hierárquico do arquivo XML desde o começo, economizando tempo e evitando algum re-trabalho necessário caso as informações do grafo estivessem inicialmente sendo inseridas no sistema manualmente.

4.2 Teste do gerador de XML

Para testar esta parte, a aplicação geradora do XML foi utilizada para construir o próprio grafo do estabelecimento escolhido (o prédio da Engenharia Elétrica), e posteriormente esse grafo foi carregado a partir do arquivo obtido utilizando o interpretador já testado anteriormente.

Primeiramente, a partir do mapa do bloco C - pavimento superior, a aplicação de geração de XML mostra a figura carregada (vide Figura 4.1), permitindo que se

indique seu nível hierárquico e seu nome. Sobre esse mapa, cada nó é caracterizado por um ponto, isto é, sua coordenada (x,y), seu andar e seu tipo (final, cruzamento, ponte ou submapa).

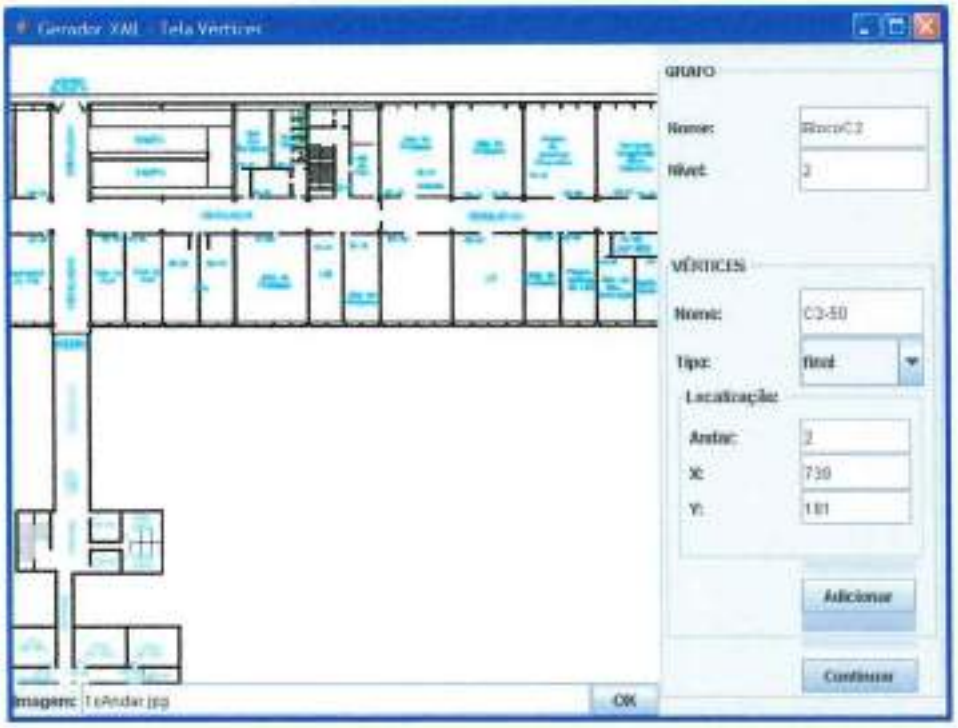


Figura 4.1 – Tela de inserção dos vértices.

A seguir, são marcadas as arestas do grafo (vide Figura 4.2), indicando os nós que estão em seus extremos e os custos daquela aresta, referentes a cada tipo de usuário.

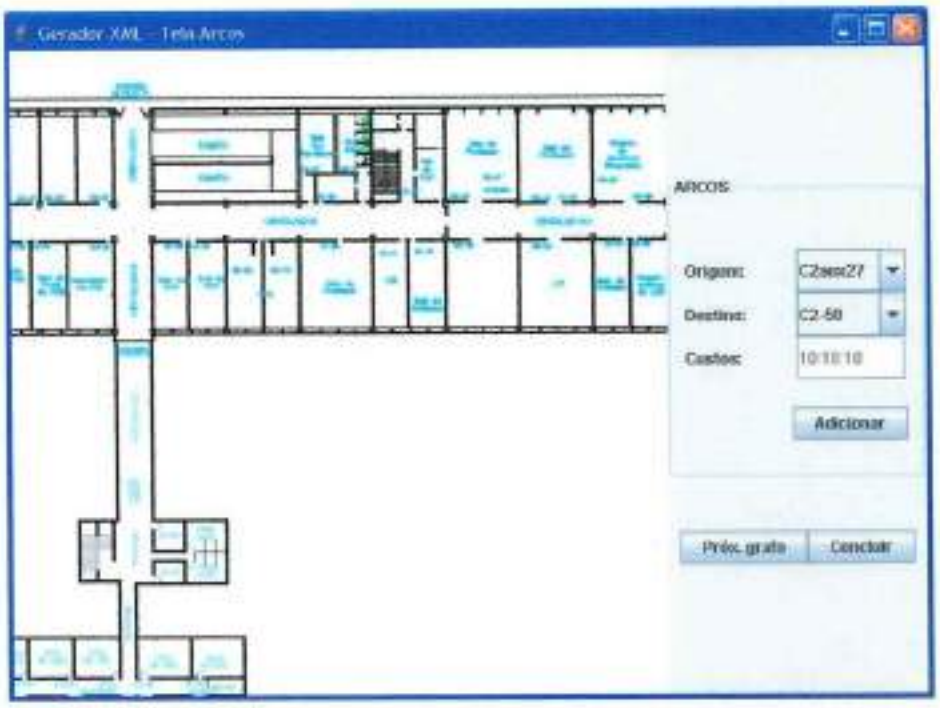


Figura 4.2 – Tela de inserção dos arcos.

O gerador de XML funcionou bem nos testes, mostrando que existiam possíveis melhorias no que se refere à interface com o usuário, uma vez que esta aplicação poderia ajudar mais na árdua tarefa de mapear os vértices e arcos do grafo sobre a figura do estabelecimento. Os pontos falhos detectados estão associados basicamente à falta de visualização durante o mapeamento dos vértices e arcos que já foram adicionados ao modelo.

4.3 Teste de comunicação cliente-servidor

Este teste visava verificar o funcionamento correto da comunicação do cliente com o servidor através do RMI. Para realizá-lo antes de possuir os outros módulos do sistema, foram criados métodos no lado servidor que não continham a parte da lógica definida para o projeto, mas sim as funções com valores de retorno de testes.

O teste consistiu na instanciação de uma classe do servidor, da realização da chamada de uma função do servidor e da utilização do valor devolvido pelo servidor. O resultado esperado deste teste é uma saída no console do servidor quando o cliente fizesse uma chamada de uma função no lado servidor.

Foi criado um script ant para ajudar a compilação e preparar o servidor para ajudar nos testes.

Os resultados obtidos para o ambiente do PDA não foram satisfatórios, sendo que este fato foi devido à máquina virtual Java Jeode não ter suporte completo ao RMI, resultando em erros devido às classes faltantes.

Após a solução do problema referente ao suporte do RMI pela máquina virtual com a substituição da JVM Jeode pela JVM CrEme no PDA, ocorreram erros na tentativa de enviar um objeto via RMI, sendo que esse erro foi facilmente resolvido implementando as classes como "serializável", que permite o empacotamento e transmissão desses objetos.

Com essas alterações o teste foi executado com sucesso e esta parte do sistema já pôde ser utilizada na implementação das outras partes.

4.4 Testes do algoritmo de roteamento

Durante a implementação deste módulo, um grafo simples de testes (de uma estrutura imaginária) foi utilizado, sendo que após o término da implementação começou-se testar o algoritmo desenvolvido num grafo maior e mais complexo. Este grafo é do estabelecimento para o qual o projeto foi desenvolvido (o Prédio da Elétrica), sendo que existem muito mais vértices e arcos além de todos os casos possíveis para testes. Com estes últimos testes foi possível obter resultados mais convincentes com relação ao tempo de resposta do algoritmo, mostrando a viabilidade do mesmo.

Com a parte cliente-servidor já funcionando, passou-se a testar o algoritmo de roteamento integrado com o resto do sistema, exceto o módulo de localização que não havia sido concluído. Para permitir que este teste fosse realizado, a parte de localização foi simulada, deixando os retornos da função de localização do servidor pré-estipulados. Com isso, desejava-se além de testar o funcionamento do algoritmo de roteamento, verificar a avaliação do applet com relação a possíveis desvios do caminho por parte do usuário (simulação da localização) e solicitação de recálculo de caminho neste caso.

Os resultados tanto do algoritmo de roteamento isoladamente como deste teste integrado e sendo chamado pelo applet foram conforme o esperado e observou-se o correto funcionamento de toda a lógica do applet em detectar quando o usuário está indo pelo caminho errado e solicitar nova rota. Este último teste do applet com o algoritmo de roteamento também obteve sucesso sendo executado com o applet rodando no PDA.

4.5 Testes do módulo de localização

Para testar o funcionamento correto dos módulos de localização, da interface do sistema com o software Ekahau e do próprio Ekahau foram realizados testes que consistiram basicamente em caminhar pelo estabelecimento e verificar se a localização estava coerente com o esperado.

Foi testado inicialmente o exemplo fornecido pelo próprio SDK do Ekahau, para entender o funcionamento e o uso deste, e para verificação da precisão.

Neste teste foi observado que depois da calibração do Ekahau através do Ekahau Manager, é necessário gravar o modelo calibrado no servidor Ekahau usando a função oferecida no menu do Ekahau Manager para conseguir obter os dados da localização através do SDK.

Outra observação realizada foi que os melhores resultados quanto à precisão da localização são obtidos quando esta é realizada utilizando a mesma marca e modelo de placa wireless com os quais foi feita a calibração do Ekahau, uma vez que o modelo calibrado se baseia nas intensidades de sinal da placa usada na calibração.

Observou-se também que, com a presença da mesma quantidade de pontos de acesso Wi-Fi, a posição destes é um fator importante para obter uma melhor precisão.

Após este teste inicial com os exemplos do próprio SDK do Ekahau, a integração do módulo de localização no sistema foi finalmente testada para verificação do funcionamento do mesmo integrado ao applet e sendo obtida através

da chamada de função ao servidor através do RMI. O objetivo do teste era verificar o funcionamento correto da interface do nosso sistema com o SDK do Ekahau.

Os resultados destes testes do módulo de localização mostraram que a precisão do Ekahau é muito dependente da marca e modelo da placa de rede e da disposição conveniente dos pontos de acesso. Com alguns modelos de placas, a localização fica mais imprecisa, porém ainda possível de se utilizar o sistema, enquanto que para outras placas de redes a localização fica totalmente aleatória, inviabilizando a utilização do sistema nestes casos.

Além disso, foi observado (vide Figura 4.3) que foram obtidos melhores resultados nos casos em que o usuário permanecia parado por cerca de 5 segundos, de forma a compensar o atraso causado pela taxa de amostragem do Ekahau.

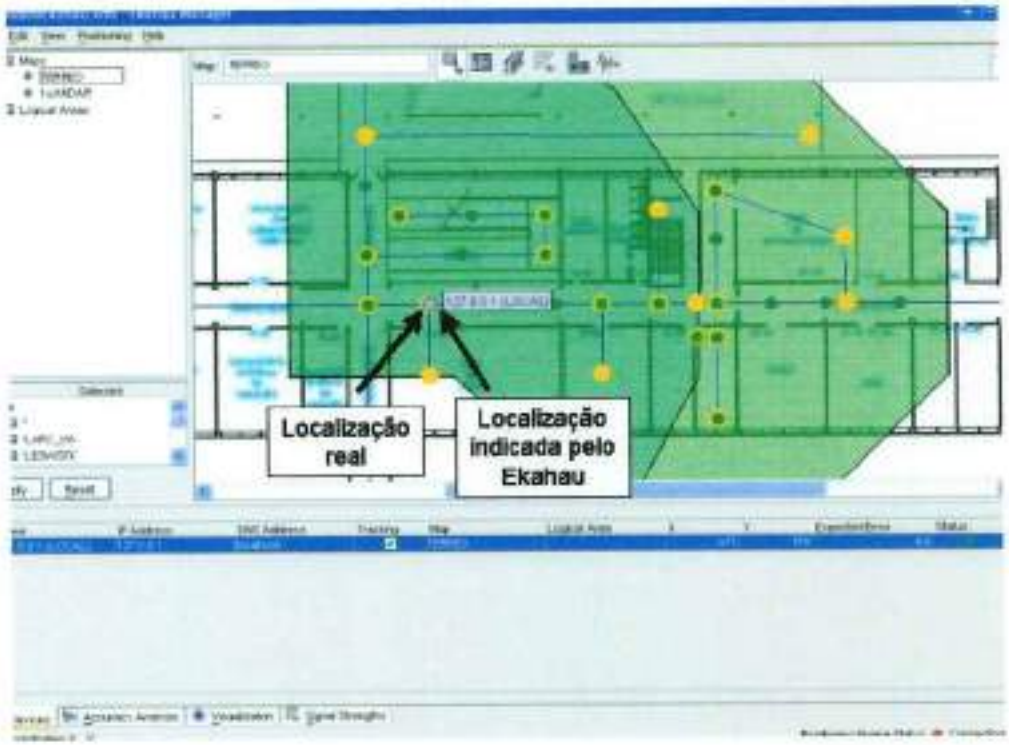


Figura 4.3 –Localização mais precisa (usuário parado)

A oscilação observada na Figura 4.4 pode ser justificada por interferências de sinais e paredes do edifício. Além disso, a disposição dos pontos de acesso encontrada no prédio não é favorável ao módulo de localização. Sua distribuição ao longo do corredor diminui a acurácia do resultado retornado pelo Ekahau.

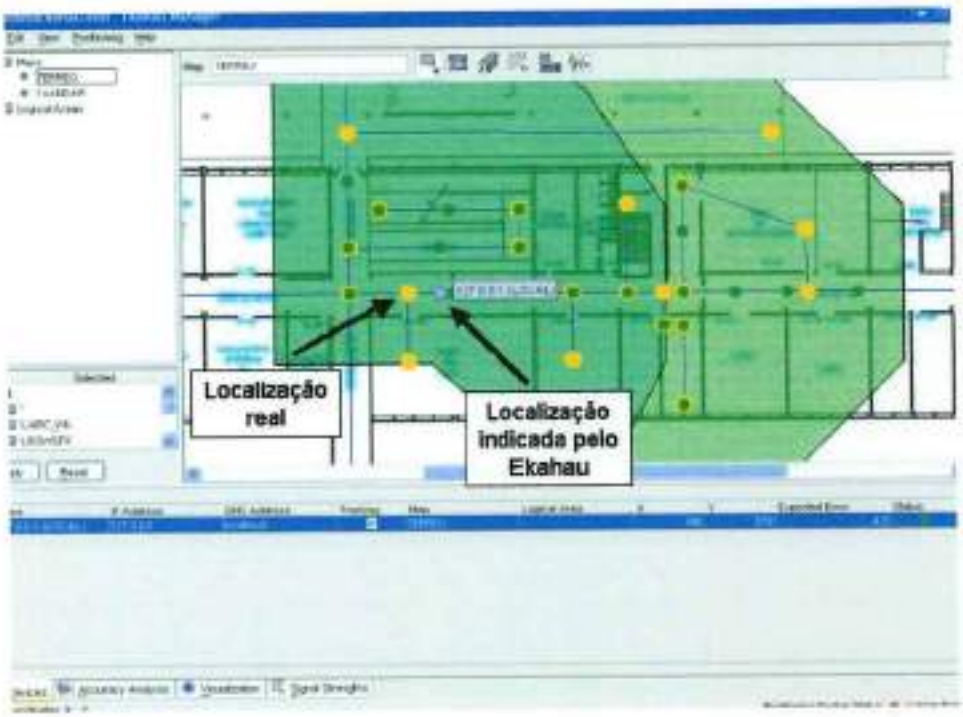


Figura 4.4 – Oscilação na localização (usuário parado)

Com o usuário em movimento, pode-se observar um atraso contínuo na localização, conforme indica a Figura 4.5.

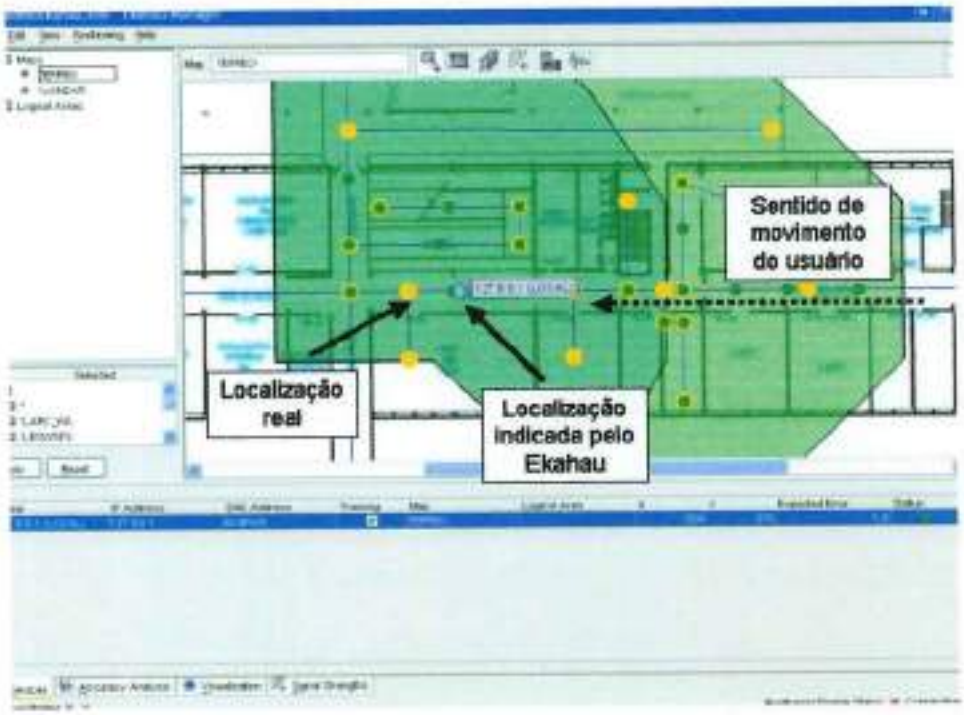


Figura 4.5 – Localização com atraso (usuário andando)

No caso do PDA, o problema apresentado foi ainda mais grave, uma vez que a placa de rede wireless interna do dispositivo disponível para desenvolver o projeto não fornecia as medidas de intensidade de sinal da maneira que o Ekahau necessita, inviabilizando sua localização por meio do software. Esse fato forçou a utilização de outros dispositivos móveis para testar o sistema, sendo que passamos a utilizar Notebook e Tablet PC nos testes.

Vale lembrar que muitos PDAs possuem placas de rede (tanto internas quanto externas) compatíveis com o Ekahau, e portanto estes dispositivos poderiam utilizar o sistema normalmente, necessitando testes mais extensivos para verificar a precisão das diversas placas de redes existentes. As marcas e modelos de placas de rede compatíveis com o Ekahau podem ser obtidas em (EKAHAU, 2005).

As marcas de placas testadas foram a D-Link, uma das marcas recomendadas pelo Ekahau, que foi utilizada na calibração do Ekahau e apresentou a melhor precisão na localização; uma placa da Intel interna ao Tablet PC, que apresentou precisão um pouco inferior, porém ainda utilizável; uma placa da 3Com, que apresentava a localização de modo totalmente aleatório, mesmo se a calibração tivesse sido realizada com a própria placa, devido à imprecisão na medida das potências dos sinais da mesma; além da placa interna do HP Pocket PC iPAQ h5550, que não fornecia os dados utilizados pelo Ekahau, conforme dito anteriormente, causando erro na localização.

4.6 Teste completo do sistema

Para testar o funcionamento completo do sistema, com todos os módulos interagindo, foi realizado um teste no próprio estabelecimento mapeado (o Prédio da Elétrica) para contemplar os diversos casos possíveis de operação. Este teste foi dividido em quatro partes:

1. A partir de um local (da frente da sala C1-46), solicitamos o caminho para chegar a outro local (sala C2-26), simulando um usuário sem nenhuma restrição. O objetivo era verificar se o caminho apresentado era o correto e percorrer o caminho corretamente verificando o funcionamento do sistema.

2. A partir do mesmo local, solicitamos o caminho para se atingir o mesmo destino com a restrição de "cadeirante". O objetivo era verificar se o caminho apresentado era diferente do anterior, sendo que para subir do primeiro para o segundo andar é apresentada a rampa como solução.

3. A partir do mesmo local, solicitamos o caminho para o mesmo destino sem nenhuma restrição. Porém ao invés de subir pela escada apresentada, seguimos pelo corredor em direção ao cruzamento do bloco D. O objetivo era observar se, conforme o usuário fosse caminhando para o lado errado, o caminho seria recalculado e apresentado novamente ao usuário, até que em determinado momento, outro caminho (pela escada do bloco D) fosse o de menor custo, sendo apresentado este novo caminho ao usuário.

4. Modelamos a escada (em frente do LARC) como uma escada rolante com sentido único para cima, fizemos a requisição de caminho do LARC para o LTS, e fizemos outra requisição de caminho do LTS para o LARC. O objetivo era comparar os caminhos gerados e comprovar a utilização de arcos direcionais.

De um modo geral, o objetivo destes testes foi verificar a exibição do caminho esperado no applet e o acompanhamento da localização com a locomoção do usuário; obter caminhos diferentes para diferentes restrições do usuário (mostrando o funcionamento dos custos variáveis dos arcos); fazer nova requisição do caminho quando o usuário não está seguindo o caminho sugerido; testar o funcionamento de arcos direcionais para alguns casos especiais como escada rolante ou rua de mão única, que não possuímos no estabelecimento adotado, mas que são suportados pelo algoritmo de roteamento.

5 CONCLUSÕES E TRABALHOS FUTUROS

É muito comum sentir a falta ou mesmo dificuldade em obter informações sobre como chegar a um dado ponto dentro de um estabelecimento como, por exemplo, um shopping center. Ainda mais complicada é a situação de pessoas que possuem algum tipo de deficiência de locomoção, como indivíduos que utilizam cadeiras de rodas.

Pensando nesses problemas, este projeto foi desenvolvido com o objetivo de tentar ajudar tanto pessoas que necessitassem de algum tipo de auxílio especial quanto pessoas comuns, melhorando a infra-estrutura para a disponibilização de informações relativas às localidades de um dado estabelecimento.

O sistema em questão tem a capacidade de encontrar a posição do usuário dentro do estabelecimento e, tendo como base suas possíveis restrições e o ponto de destino procurado por ele, indicar um caminho que seja otimizado para atender as suas necessidades e possibilidades.

Durante seu desenvolvimento, foram encontradas diversas dificuldades, que estavam relacionadas principalmente à localização do usuário. Problemas como incompatibilidade entre dispositivos, infra-estrutura de rede disponível deficitária para as necessidades do projeto e utilização de um software comercial provocaram alguns atrasos no projeto. Apesar disso, o projeto conseguiu cumprir suas metas e seu cronograma.

O sistema final desenvolvido não permite que um grande número de usuários se conectem ao sistema simultaneamente devido às limitações da versão *trial* do Ekahau, nem que um usuário conectado se locomova em alta velocidade dentro do estabelecimento sem que haja um atraso significativamente alto na sua localização, restrições estas impostas por algumas das características citadas no parágrafo anterior. Além disso, o sistema não é capaz de atender as necessidades de usuários com deficiências visuais.

Entretanto, foi possível atingir resultados bastante satisfatórios na conclusão deste trabalho. Apesar de uma certa imprecisão na localização obtida com o sistema, este consegue descobrir de forma aceitável a posição do usuário e traçar a melhor

rota a ser percorrida por ele, atingindo portanto o objetivo de auxiliá-lo em sua locomoção dentro de um estabelecimento.

Como lições aprendidas, podemos citar a importância de construir sistemas com alto grau de modularidade, permitindo a modificação do conteúdo de cada módulo sem que o sistema seja afetado com as alterações realizadas. Adicionalmente, percebemos a grande dificuldade em utilizar softwares comerciais, pois informações relativas ao modo como foram construídos são muito escassas, impedindo que sejam realizadas adaptações no projeto.

5.1 Trabalhos Futuros

A partir da idéia desenvolvida neste projeto, existem diversas possibilidades de mudanças, otimizações e inclusões que podem ser trabalhadas futuramente, tornando o projeto mais abrangente e completo.

5.1.1 Desenvolvimento do módulo de localização

Para o módulo de localização do sistema, optou-se por utilizar a versão "demo" do software comercial Ekahau. Esta decisão foi justificada pelo fato de que desenvolver tal módulo não foi o enfoque principal deste projeto. Adicionalmente, não havia tempo disponível para obter conhecimentos mais profundos no assunto em questão, visto que o enfoque deste projeto estava no desenvolvimento do algoritmo de roteamento.

O desenvolvimento futuro do módulo de localização proporcionaria a independência de softwares comerciais, diminuindo o custo do projeto devido a licenças. Além disso, seria possível eliminar as limitações impostas pelo software "demo", como o número de dispositivos a serem rastreados pelo Ekahau Engine. Finalmente, o desenvolvimento de um aplicativo próprio permitiria sua otimização para o projeto, gerando resultados mais precisos e diminuindo o tempo de atraso no fornecimento da localização do usuário.

5.1.2 Representação das coordenadas de localização

O sistema de coordenadas adotado no projeto considera cada ponto como sendo um par (x, y), representando a posição do usuário em um determinado mapa, e uma terceira coordenada indicando o andar em que o usuário está. No caso deste projeto, o sistema de coordenadas proposto não sofre problemas de imprecisão, pois cada um dos andares possui um único mapa que interliga os diferentes blocos do estabelecimento. Entretanto, o modelo adotado perde eficácia nos casos de implementações em locais onde existam pontos de coordenadas equivalentes, ou seja, existem mapas contendo pontos diferentes, porém representados pelas mesmas coordenadas (x, y, andar), provocando a falta de unicidade.

A solução de tal problema poderia ser uma mudança na representação das coordenadas, sendo que, ao invés de indicar somente o andar ao qual um dado ponto pertence, seria acrescentada a informação de qual mapa contém o ponto, evitando assim as ambigüidades citadas anteriormente.

5.1.3 Interface com o usuário

Ainda tendo como motivação a preocupação em atender os diversos tipos de necessidades existentes entre os usuários deste sistema, novos recursos de interface homem-máquina poderiam ser desenvolvidos com o objetivo de atender a deficientes visuais, ou somente para tornar mais fácil a compreensão do caminho fornecido pelo sistema.

A primeira proposta atenderia os usuários do sistema que possuíssem deficiências visuais. Todos os dados obtidos atualmente através do site poderiam ser capturados por aplicativos de reconhecimento de comandos de voz, e seriam processados normalmente pelo sistema. Para retornar ao usuário o caminho calculado, o sistema produziria instruções audíveis capazes de orientá-lo por todo o trajeto a ser percorrido, incluindo instruções de correção de rota no caso de erros de percurso.

A outra proposta sugere a utilização de animações 3D para mostrar o caminho que deve ser percorrido. O dispositivo cliente poderia simular o ponto de vista do

usuário, tornando a interface muito mais atraente e amigável, além de facilitar o reconhecimento por parte do usuário do local que está sendo explorado.

5.1.4 Análise de perfil dos usuários

O sistema desenvolvido neste projeto poderia ser utilizado em diferentes tipos de estabelecimentos, entre eles hospitais, centros de exposição e shopping centers. No caso destes dois últimos, o armazenamento do perfil dos usuários poderia ser bastante interessante, pois as informações obtidas poderiam ser utilizadas, por exemplo, para a escolha na divulgação de propagandas mais atraentes a cada usuário. Além disso, o sistema poderia disparar informações de um dado ponto quando um usuário se aproximasse dele.

5.1.5 Unificação das fontes de dados

Para o armazenamento dos dados disponibilizados no site do sistema, foi utilizado um banco de dados, enquanto que os dados relativos ao grafo foram armazenados em um arquivo XML. Por se tratarem de dados bastante próximos nos dois casos, seria interessante que eles tivessem sido armazenados em uma única fonte, evitando assim o re-trabalho e possíveis inconsistências.

ANEXO A – CASOS DE USO

Nesta seção é apresentado o diagrama de casos de uso com sua respectiva descrição, a fim de entender os requisitos funcionais do sistema (vide Figura A.1). Estes dados foram muito utilizados na fase de implementação, principalmente em sua validação.

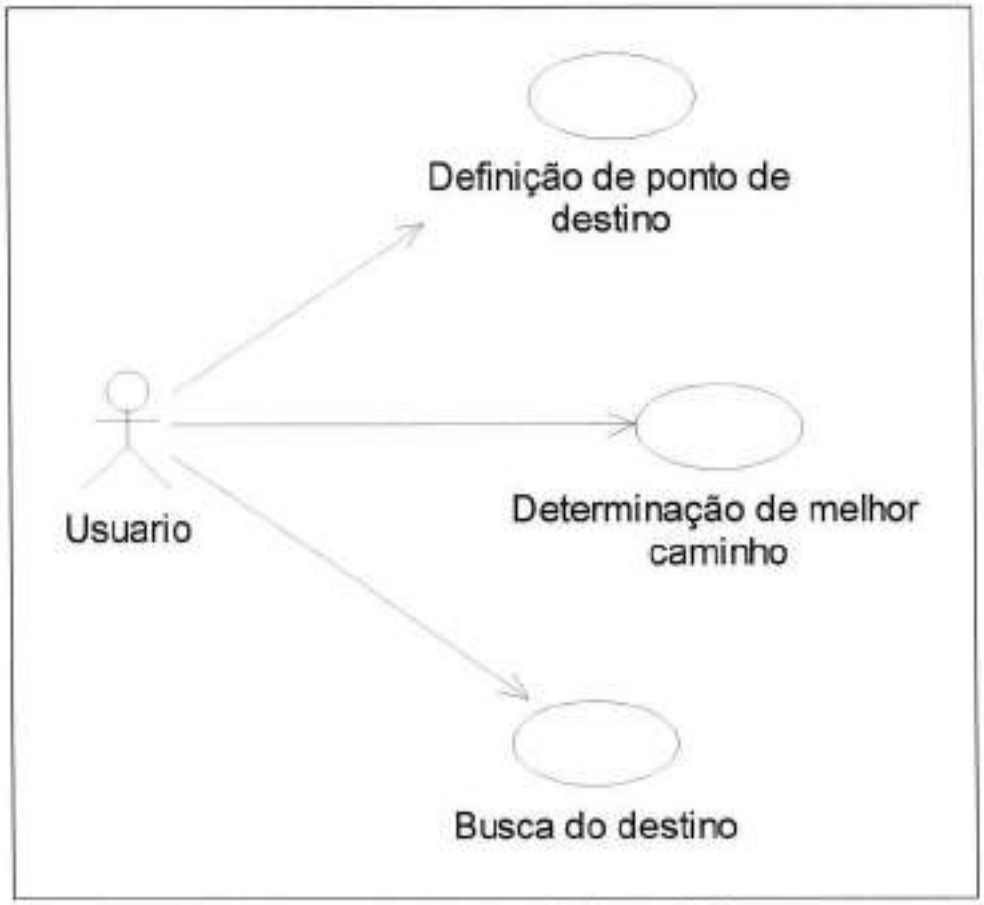


Figura A.1 - Diagrama de Casos de Uso.

Caso de uso 1: Definição de Ponto de Destino

Descrição: Este caso de uso descreve o processo para a entrada do ponto de destino no sistema.

Evento iniciador: Usuário ordena busca por um ponto do estabelecimento.

Atores: Usuário.

Pré-condição: Usuário dentro do estabelecimento.

Seqüência de eventos:

1. Sistema apresenta grupos de locais de destino;
2. Usuário escolhe um dos itens;
3. Sistema detalha item escolhido;
4. Usuário escolhe ponto de destino (uma das folhas da árvore de localização);
5. Sistema define o ponto escolhido como nó de destino.

Pós-condição: Ponto de destino foi inserido no sistema.

Extensões: Não há extensões.

Inclusão: Não há inclusões.

Caso de uso 2: Determinação do Melhor Caminho

Descrição: Este caso de uso descreve o processo para a determinação do melhor caminho entre dois pontos localizados no estabelecimento.

Evento iniciador: Usuário ordena busca do melhor caminho.

Atores: Usuário

Pré-condição: Ponto de destino definido.

Seqüência de eventos:

1. Sistema localiza usuário dentro do estabelecimento através da rede wireless;
2. Sistema define o ponto de localização do usuário como nó de partida;
3. Sistema faz o roteamento através de análise de custos dos caminhos;
4. Sistema apresenta resultado ao usuário.

Pós-condição: Melhor caminho entre dois pontos do estabelecimento foi determinado.

Extensões: Não há extensões.

Inclusão: Não há inclusões.

Caso de uso 3: Busca do destino

Descrição: Este caso de uso descreve o processo para a busca do local de destino por nome da loja

Evento iniciador: Usuário pressiona o botão de confirmação

Atores: Usuário

Pré-condição: Destino preenchido

Seqüência de eventos:

1. Sistema verifica o nome do destino preenchido
2. Sistema mostra os resultados que mais se aproximam da consulta realizada
3. Usuário seleciona o destino desejado dentro da lista gerada
4. Sistema exibe o mapa

Pós-condição: Melhor caminho entre dois pontos do estabelecimento foi determinado.

Extensões: Destino não preenchido

Inclusão: Não há inclusões.

ANEXO B – MODELO DE CLASSES

Nesta seção são apresentados os diagramas de classes com a especificação de seus atributos e métodos. Estes dados foram muito utilizados na fase de implementação do projeto, e serviram de guia para o desenvolvimento de todos os módulos do sistema.

A arquitetura utilizada neste projeto é a cliente-servidor, pois uma parte do sistema estará rodando no PDA do usuário (em forma de applet) enquanto que a parte central do sistema fica no servidor central (vide Figura B.1). Deseja-se que a parte cliente consuma o mínimo processamento e memória possíveis, devido às restrições do dispositivo móvel.

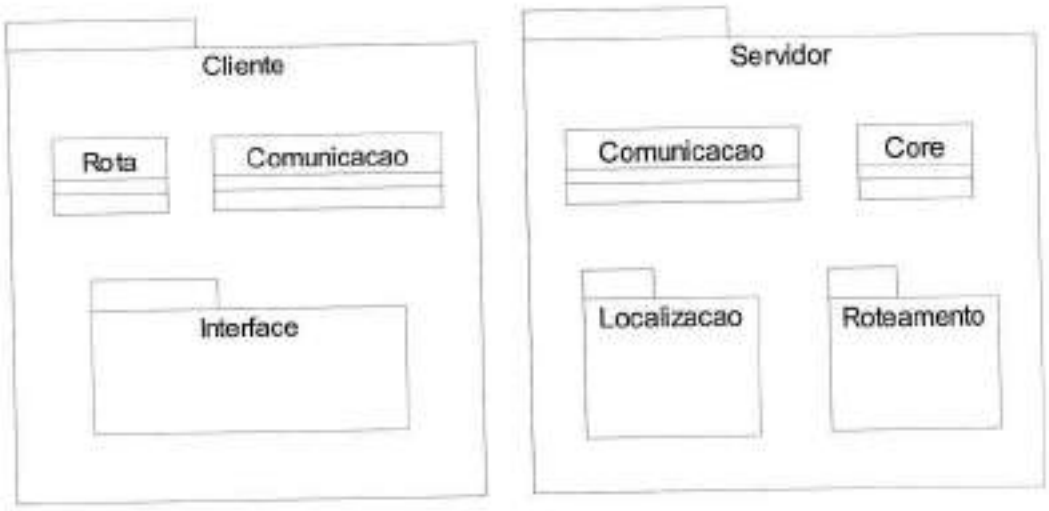


Figura B.1 – Diagrama de classes: Arquitetura cliente-servidor.

O pacote cliente consiste de uma interface que interage com o cliente para recolher os dados do cliente e também representar as respostas do sistema para o cliente. Os dados a serem recolhidos por este pacote seriam o destino que o usuário almeja. A resposta do sistema é representada de forma gráfica com a possibilidade de ampliar e diminuir o mapa de forma adequada pelo sistema ou pelos comandos do usuário.

O pacote servidor tem as funções de localizar o usuário e calcular um caminho mais adequado para o usuário, conforme as suas necessidades. O usuário é localizado

através das intensidades dos sinais recebidos pelos "access points" no módulo de localização. Com os dados do posicionamento do usuário e o destino informado pela interface do pacote cliente, o módulo de roteamento determina uma rota para guiar o usuário.

I. Módulo de Roteamento

O módulo de roteamento é o mais complexo do sistema, portanto será apresentada a seguir uma descrição detalhada de suas classes, métodos e atributos, ilustrados na Figura B.2.

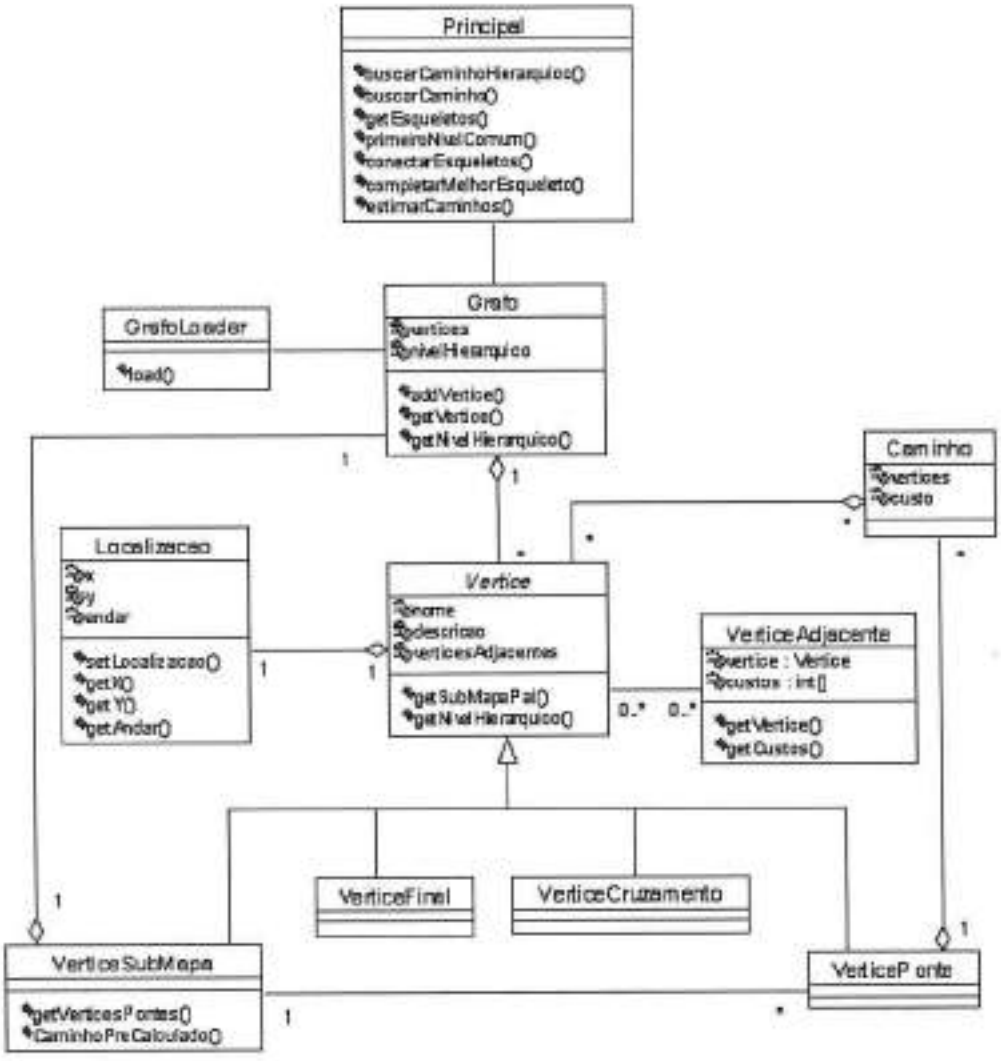


Figura B.2 – Diagrama de Classes: Módulo de Roteamento.

Classe Principal

Descrição: Classe responsável pela busca do caminho. Também tem a função de interface com os outros módulos do sistema.

Atributos: Não se aplica

Métodos:

- buscarCaminhoHierarquico: Método responsável pela busca do caminho utilizando o grafo hierárquico.
    - entrada: vértice inicial e vértice final.
    - saída: caminho encontrado.
- buscarCaminho: Método responsável pela busca do caminho quando o vértice inicial e o final estão no mesmo submapa.
    - entrada: vértice inicial e vértice final.
    - saída: caminho encontrado.
- getEsqueletos: Método responsável por criar o esqueleto de caminho com as conexões entre os diferentes níveis da hierarquia. Submapas são conectados aos seus submapas pais através dos vértices pontes.
    - entrada: vértice inicial e nível hierárquico de destino.
    - saída: esqueleto de caminho.
- PrimeiroNivelComum: Este método encontra o primeiro submapa que inclui ou contém os vértices inicial e destino num nível superior da hierarquia e retorna este nível.
    - entrada: vértices inicial e destino.
    - saída: nível da hierarquia.
- conectarEsqueletos: Método responsável por conectar dois conjuntos de esqueletos de caminho.
    - entrada: dois conjuntos de esqueletos de caminho.
    - saída: conjunto de esqueletos conectado.
- completarMelhorEsqueleto: Método responsável por completar um esqueleto, convertendo-o em um caminho completo.
    - entrada: esqueleto de caminho.
    - saída: caminho completo.

- estimarCaminhos: Este método inicializa o conjunto de esqueletos de caminho que irão conectar uma seqüência de submapas através dos níveis de hierarquia. Os custos dos caminhos são aproximados com uma função heurística baseada na distância euclidiana.
    - entrada: o vértice inicial e um conjunto de vértices.
    - saída: conjunto de esqueletos de caminhos com os custos dados pela função heurística.

Classe Grafo

Descrição: Esta classe representa um grafo do sistema, sendo que este representa um conjunto de vértices pertencentes a um mesmo nível hierárquico.

Atributos:

- vertices: Conjunto dos vértices pertencentes a este grafo.
- nivelHierarquico: Nível da hierarquia ao qual o grafo faz parte.

Métodos:

- addVertice: Este método visa a inclusão de um novo vértice no grafo.
    - entrada: vértice a ser incluído.
    - saída: não se aplica.
- getVertice: Este método retorna o vértice solicitado.
    - entrada: identificador do vértice.
    - saída: vértice desejado.
- getNivelHierarquico: Obtém o nível da hierarquia ao qual o grafo pertence.
    - entrada: não se aplica.
    - saída: nível da hierarquia ao qual o grafo pertence.

Classe GrafoLoader

Descrição: Esta classe realiza a função de leitura do arquivo de XML contendo as informações do grafo e a respectiva interpretação do mesmo, transformando a descrição do XML em objetos do sistema. Esta classe também visa o desacoplamento da lógica de carregamento do grafo para que seja possível a utilização de qualquer outra representação, e não somente XML.

Atributos: Não se aplica

Métodos:

- load: Este método é o responsável por carregar o grafo
  - entrada: nome do arquivo.
  - saída: grafo completo baseado na descrição do arquivo.

Classe Vertice

Descrição: Esta classe representa um vértice do sistema, contendo a descrição do mesmo e as ligações deste com os demais vértices do sistema.

Atributos:

- nome: nome do vértice.
- descricao: descrição do vértice, utilizada necessariamente nos vértices finais e opcionalmente nos demais tipos de vértices.
- verticesAdjacentes: conjunto dos vértices aos quais o vértice possui conexão diretamente com o respectivo custo.

Métodos:

- getSubMapaPai: Este método obtém o vértice do tipo submapa em que este vértice está incluído. No caso de este estar no nível 0 da hierarquia (raiz do grafo hierárquico) é retornado nulo.
  - entrada: não se aplica.
  - saída: vértice do tipo submapa no qual o vértice em questão está incluído.
- getNivelHierarquico: Obtém o nível da hierarquia ao qual o vértice pertence.
  - entrada: não se aplica.
  - saída: nível da hierarquia ao qual o grafo pertence.

Classe VerticeSubMapa

Descrição: Esta classe é uma especialização da classe Vertice e representa os vértices do tipo submapa do sistema.

Atributos: Não se aplica.

Métodos:

- getVerticesPontes: Obtém os vértices pontes deste submapa, que são os vértices que conectam o submapa ao seu submapa pai.

- entrada: não se aplica.
- saída: vértices ponte do submapa.

- caminhoPreCalculado: Obtém um caminho pré-calculado entre dois vértices, retornando nulo caso não exista um caminho pré-calculado para estes vértices.
  - entrada: vértice inicial e destino.
  - saída: caminho pré-calculado para alcançar o vértice destino a partir do inicial.

Classe VerticeFinal

Descrição: Esta classe é uma especialização da classe Vertice e representa os vértices do tipo final do sistema, que são os vértices que o usuário pode escolher como destino.

Atributos: Não se aplica.

Métodos: Não se aplica.

Classe VerticeCruzamento

Descrição: Esta classe é uma especialização da classe Vertice e representa os vértices do tipo cruzamento do sistema, que são sub-alvos utilizados para representar mudanças de direção ou cruzamentos.

Atributos: Não se aplica.

Métodos: Não se aplica.

Classe VerticePonte

Descrição: Esta classe é uma especialização da classe Vertice e representa os vértices do tipo ponte do sistema. Vértices deste tipo são vértices especiais que conectam um submapa ao seu submapa pai.

Atributos: Não se aplica.

Métodos: Não se aplica.

Classe Localizacao

Descrição: Esta classe é utilizada para representar a localização de um vértice em

relação as coordenadas cartesianas (x e y) do mapa e o andar. É útil também para desacoplar a representação da localização de um vértice do sistema, facilitando a inclusão de novos atributos e métodos no caso de serem necessárias outras informações a respeito de um vértice para algum estabelecimento especifico.

Atributos:

- x: coordenada cartesiana x.
- y: coordenada cartesiana y.
- andar: andar do estabelecimento.

Métodos:

- setLocalizacao: Ajusta a localização.
  - entrada: coordenadas (x e y) e andar.
  - saída: não se aplica.
- getX:
  - entrada: não se aplica.
  - saída: coordenada x.
- getY:
  - entrada: não se aplica.
  - saída: coordenada y.
- getAndar:
  - entrada: não se aplica.
  - saída: andar do estabelecimento.

Classe VerticeAdjacente

Descrição: Esta classe é utilizada pela classe Vertice para representar os vértices alcançáveis e o respectivo custo.

Atributos:

- vertice: vértice alcançável.
- custos: vetor de custos, sendo que cada posição representa o custo para alcançar este vértice para uma determinada restrição.

Métodos:

- getVertice: Obtém o vértice alcançável.
  - entrada: não se aplica.

- saída: vértice.

- getCusto: Obtém o custo para travessia com uma determinada restrição
  - entrada: identificador da restrição.
  - saída: custo para a travessia.

Classe Caminho

Descrição: Esta classe representa um caminho para se chegar a um nó destino a partir de um nó de origem.

Atributos:

- vertices: conjunto de vértices ordenados que representa o caminho a ser percorrido.
- custo: custo para se percorrer este caminho.

Métodos:

- addVertice: Método responsável pela adição de um vértice ao caminho.
  - entrada: vértice a ser adicionado.
  - saída: não se aplica.
- getVertices: Método utilizado para obter todos os vértices pertencentes ao caminho.
  - entrada: não se aplica.
  - saída: conjunto de vértices.
- getCusto: Obtém o custo de travessia deste caminho.
  - entrada: não se aplica.
  - saída: custo de travessia.

2. *Módulo de Localização*

Este módulo é responsável pela localização do usuário, consistindo de uma classe interfaceLBS e um pacote LBS, conforme ilustra a Figura B.3.

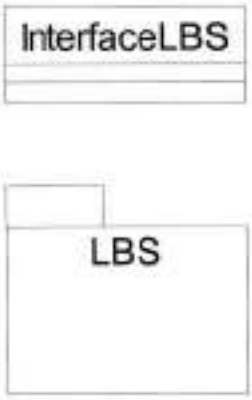


Figura B.3 – Diagrama de Classes: Módulo de Localização.

A classe interfaceLBS contém os métodos intermediários entre o sistema e o pacote LBS criando assim uma interface independente do pacote de LBS para possíveis extensões como, por exemplo, a troca do pacote LBS.

O pacote LBS inicialmente será o pacote comercial chamado Ekahau, que usa tanto as intensidades recolhidas pelos "access points" como um banco de dados das amostras pré-configuradas para melhorar o desempenho de localização. O Ekahau é um servidor de posicionamento puramente de software que tem uma acurácia em média de 1 metro baseado na tecnologia de modelo de posição. Ele oferece uma interface Java para integrar com as aplicações Wi-Fi e trabalha com as placas de rede sem fio e "access points" de padrão industrial IEEE 802.11a, 802.11b e 802.11g.

3. Módulo de Interface com o Usuário

Este módulo oferece uma interface para obter os dados informados pelo usuário (vide Figura B.4). A interface do usuário consiste em um menu inicial para o usuário escolher o destino e um mapa interativo com a rota calculada pelo sistema.

Figura B.4 – Diagrama de Classes: Módulo de Interface.

Esta parte interativa foi desenvolvida em applet, uma interface amplamente utilizada para interagir com os usuários nas aplicações cliente-servidor. Dentro deste há uma barra de ferramentas com os botões para chamar as funções e um mapa para representar a rota de forma interativa, conforme a posição atual do usuário.

ANEXO C – MAPAS DO AMBIENTE DE TESTES

A implementação deste trabalho foi realizada para o Prédio da Engenharia Elétrica da POLI, que é constituído por quatro blocos principais. Entretanto, foram considerados somente alguns de seus blocos, e em alguns casos somente parte deles, pois o mapeamento do prédio foi realizado de forma a atender os requisitos mínimos que permitissem a funcionalidade do sistema.

O bloco B é o bloco frontal, sendo que seu nível térreo pode ser visto na Figura C.1e seu nível superior, na Figura C.2.

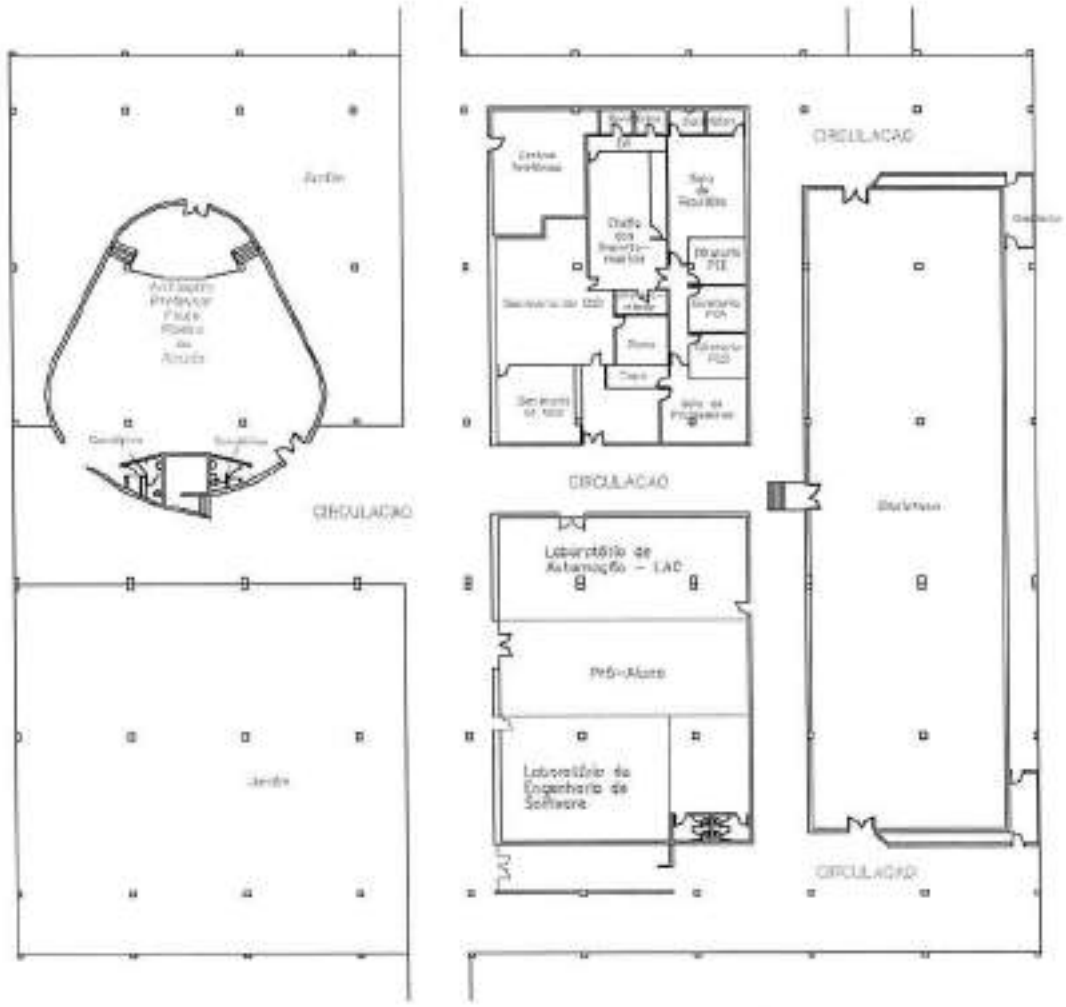


Figura C.1 – Mapa do bloco B: andar térreo.

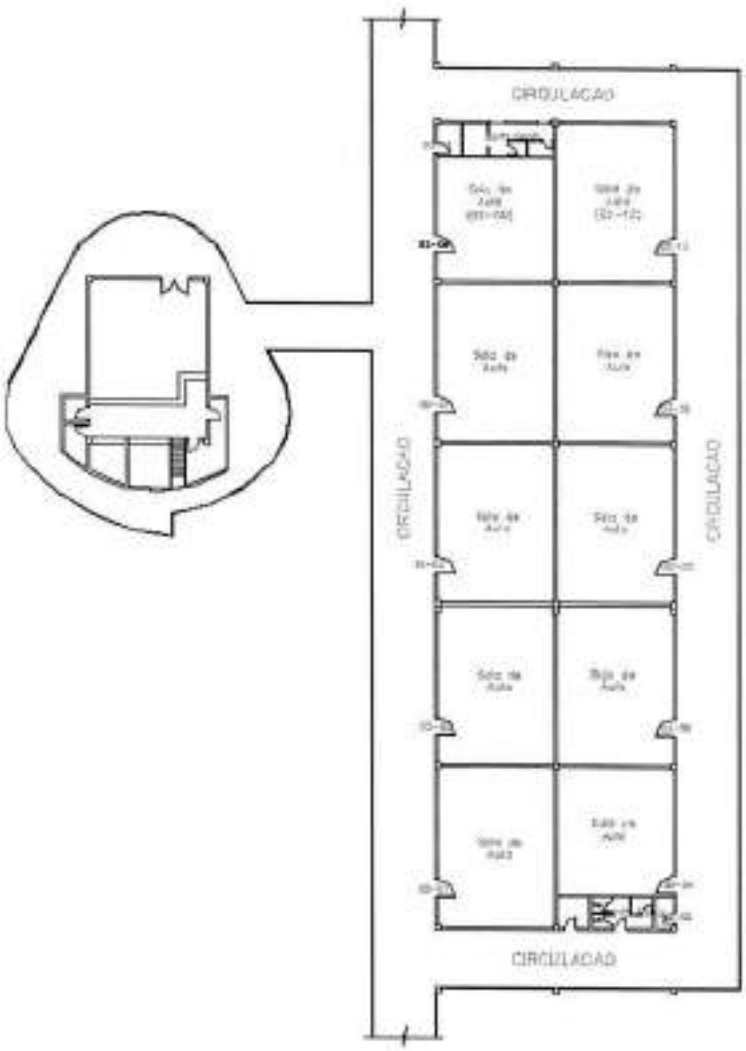


Figura C.2 – Mapa do bloco B: andar superior.

O bloco C, que se encontra à direita do bloco B, tem suas plantas indicadas na Figura C.3. O mezanino não foi considerado no mapeamento do prédio para este trabalho para que não fosse necessária a utilização de um novo mapa para representar aquele andar separadamente.

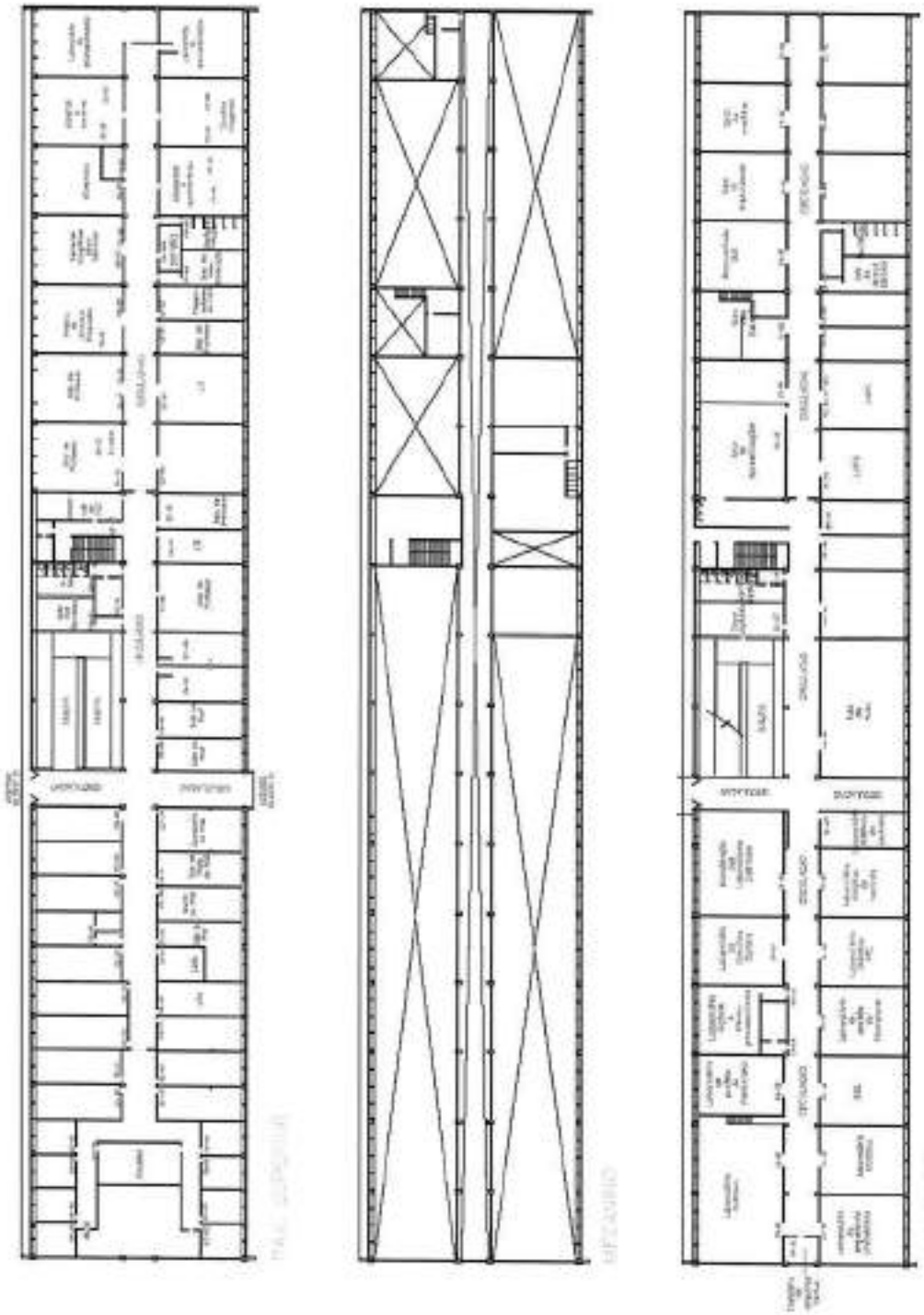


Figura C.3 – Mapa do bloco C: à esquerda, tem-se o pavimento superior; no centro, o mezanino; à direita, tem-se o pavimento térreo.

O bloco D encontra-se à direita do bloco C, sendo que seu pavimento térreo está indicado na Figura C.4, enquanto que seu pavimento superior está indicado na Figura C.5.

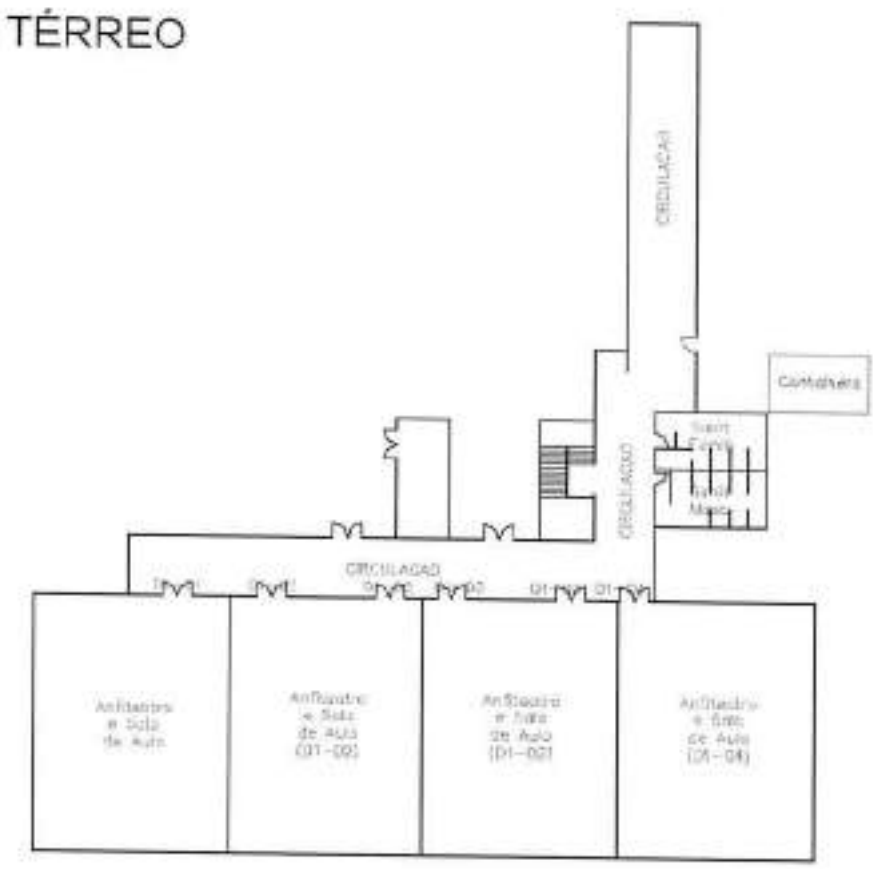


Figura C.4– Mapa do bloco D: andar térreo.

PRIMEIRO PAVIMENTO

Figura C.5 – Mapa do bloco D: andar superior.

ANEXO D – GRAFOS DO AMBIENTE DE TESTES

Nesta seção são apresentados os grafos utilizados no projeto. Na Figura D.1, pode-se observar o grafo que representa o Prédio da Elétrica. Cada nó indicado por um quadrado claro representa um submapa, ou seja, um pavimento de um determinado bloco do edifício. Os nós numerados com 1 indicam o andar térreo, enquanto que os numerados com 2 indicam o andar superior. Já os quadrados escuros indicam os nós-cruz, que representam cruzamentos.

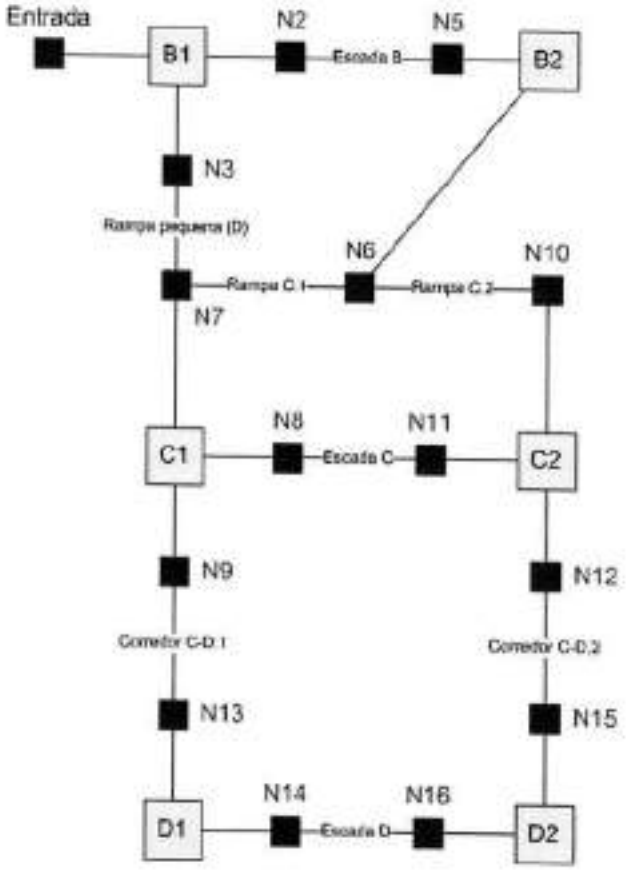


Figura D.1 – Grafo do Prédio da Elétrica.

O detalhamento dos submapas do Bloco B podem ser vistos na Figura D.2 (pavimento térreo) e na Figura D.3 (pavimento superior).

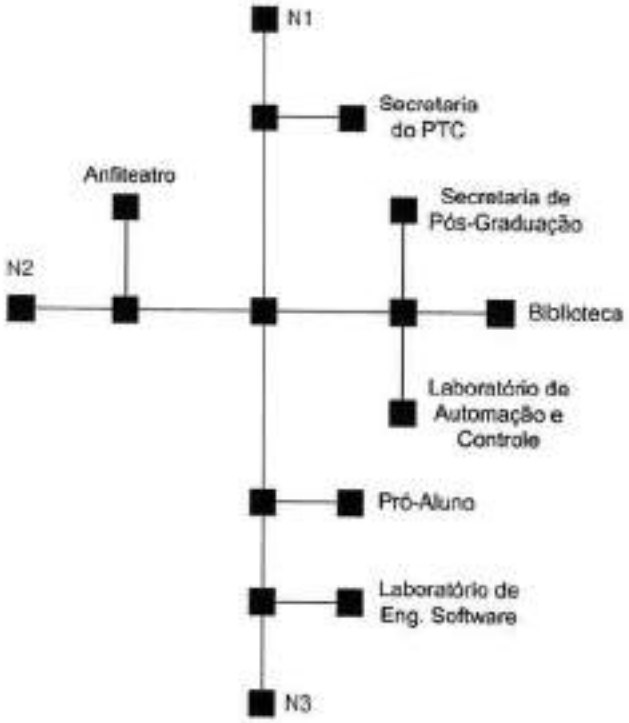


Figura D.2 – Grafo do bloco B: andar térreo (B1).

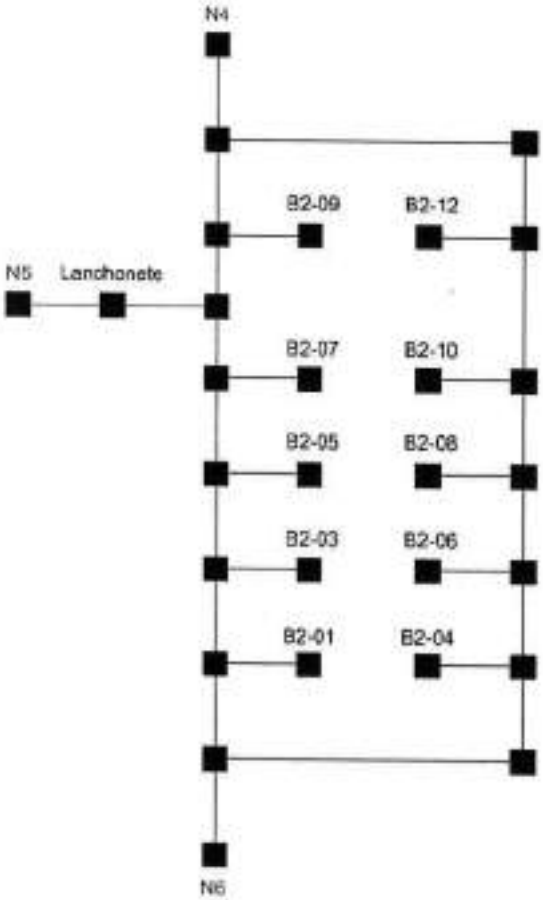


Figura D.3 – Grafo do bloco B: andar superior (B2).

O detalhamento dos submapas do Bloco C podem ser vistos na Figura D.4 (pavimento térreo) e na Figura D.5 (pavimento superior).

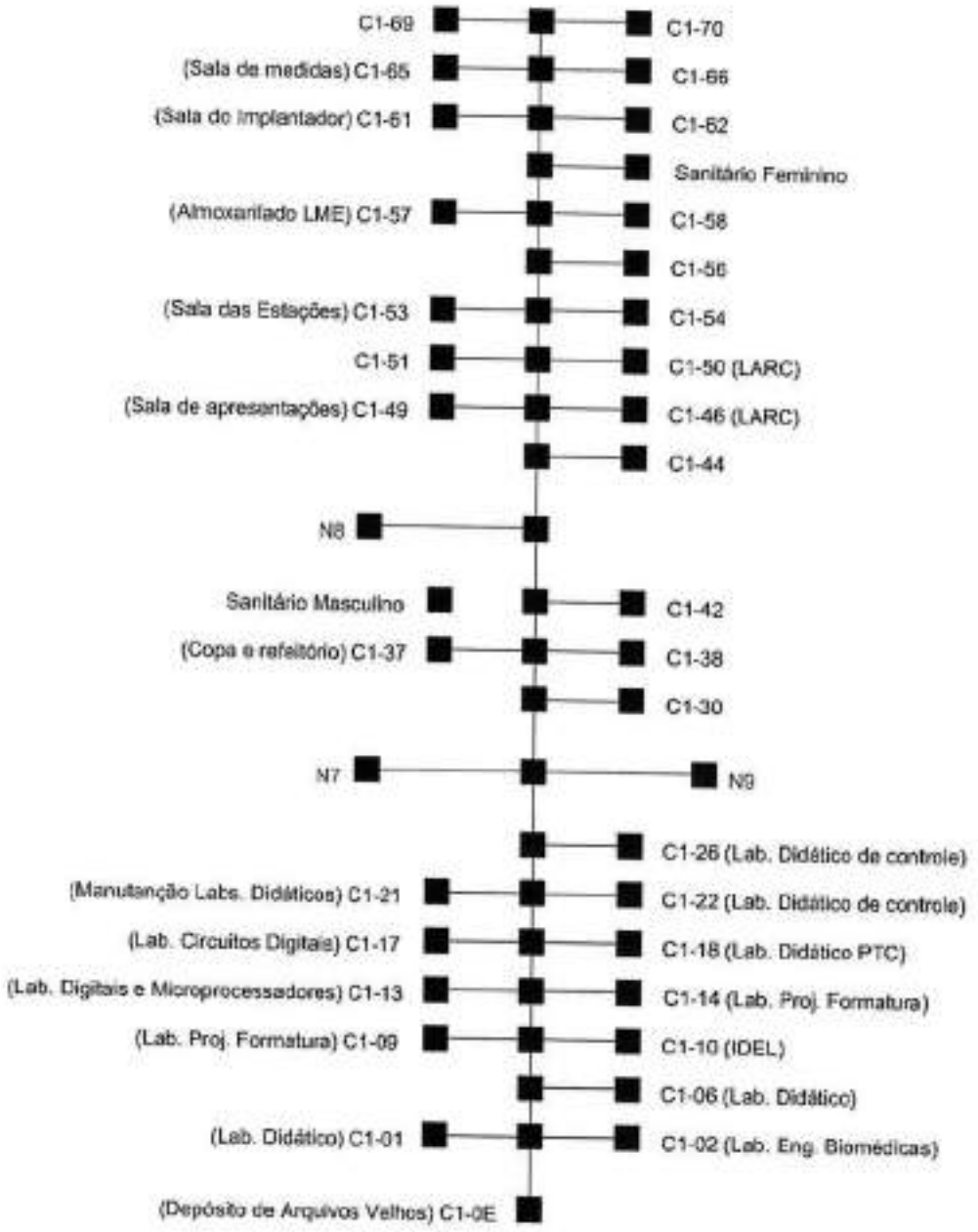


Figura D.4 – Grafo do bloco C: andar térreo (C1).

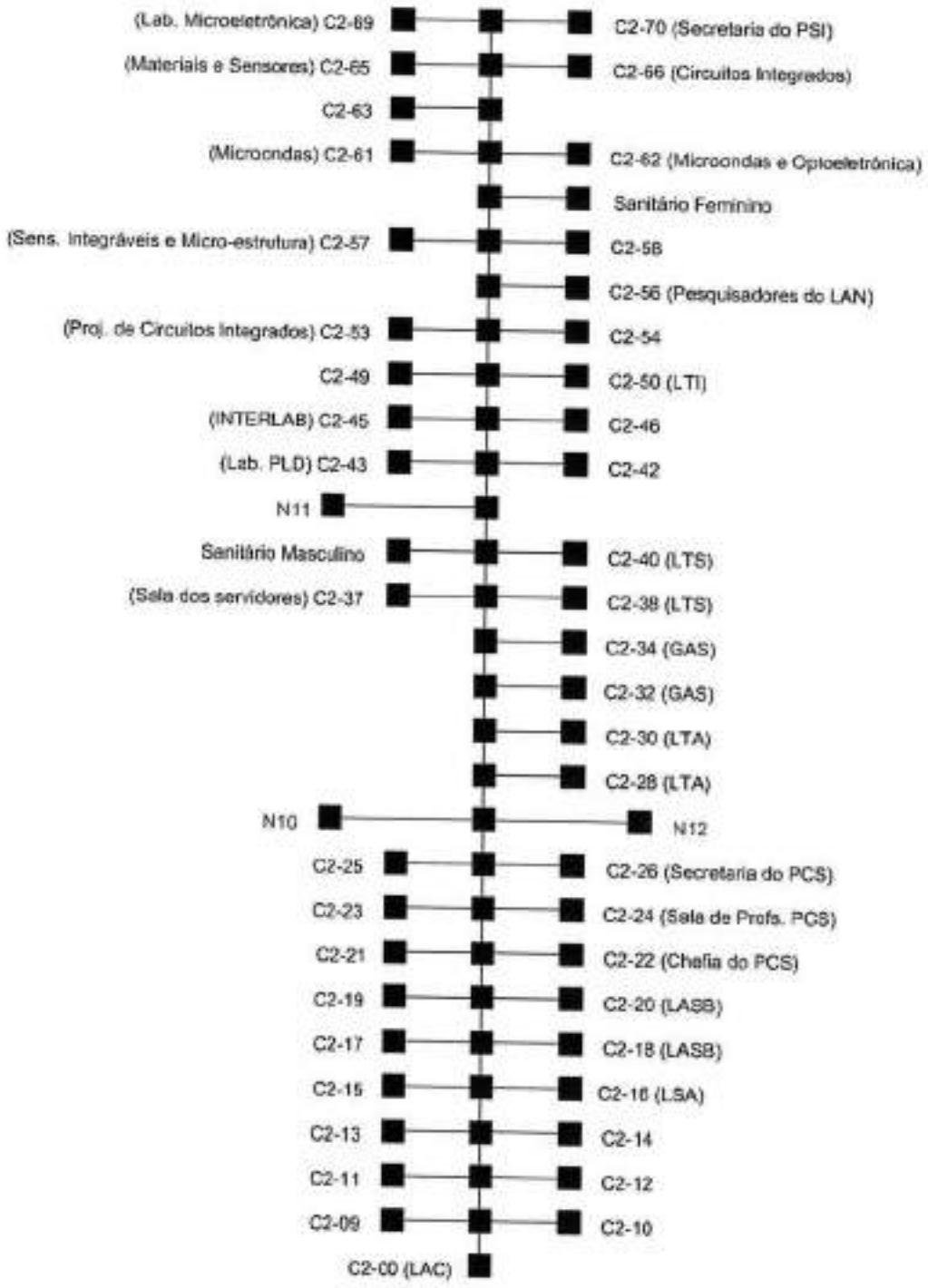


Figura D.5 – Grafo do bloco C: andar superior (C2).

O detalhamento dos submapas do Bloco D podem ser vistos na Figura D.6 (pavimento térreo) e na Figura D.7 (pavimento superior).

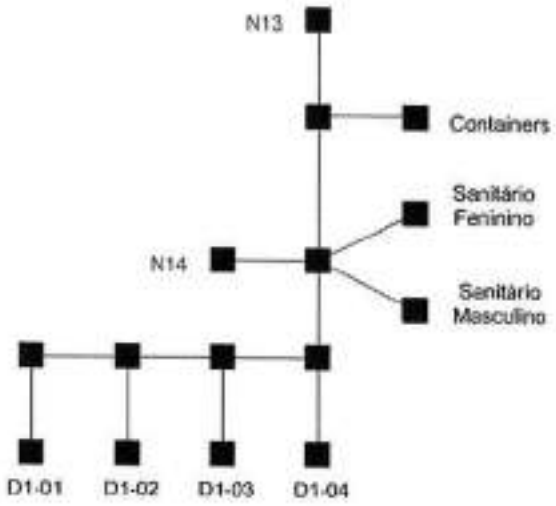


Figura D.6 – Grafo do bloco D: andar térreo (D1).

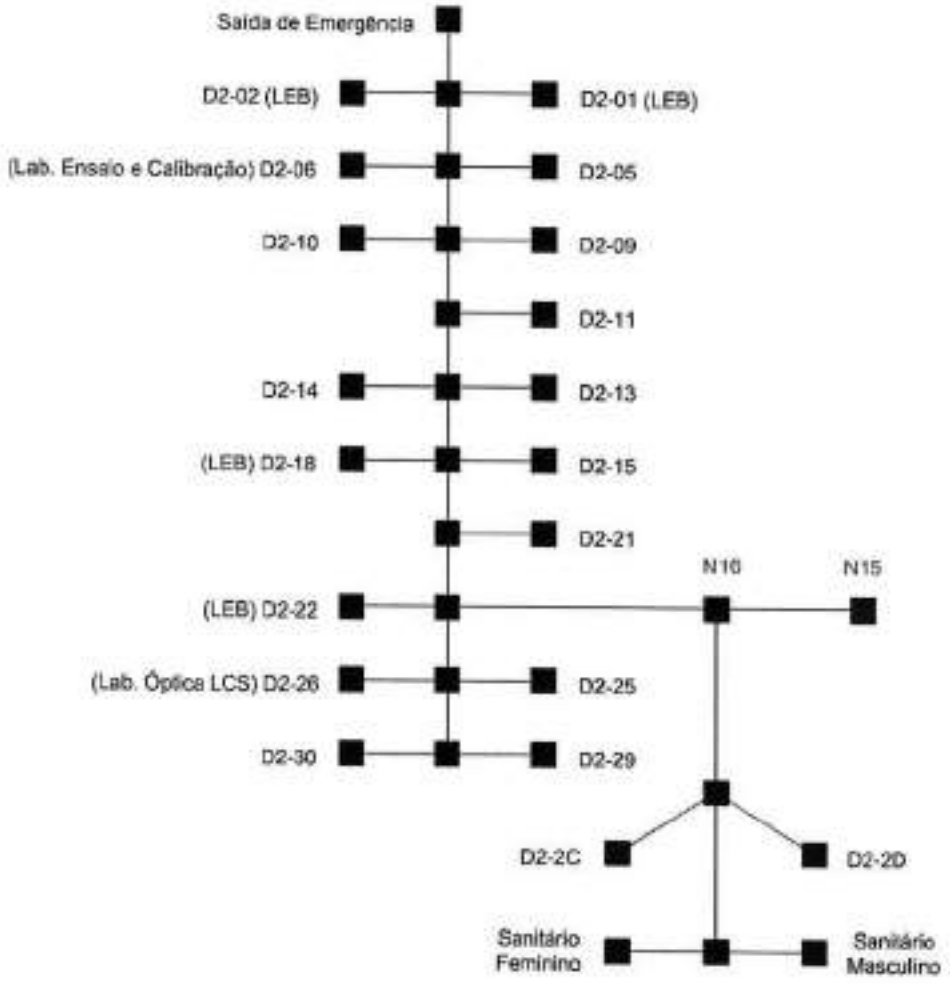


Figura D.7 – Grafo do bloco D: andar superior (D2).

ANEXO E – CÓDIGO FONTE DO ALGORITMO DE ROTEAMENTO

Nesta seção é apresentado o pseudo-código do algoritmo de roteamento desenvolvido neste trabalho.

```
buscarCaminhoHierarquico (vértice inicial, vértice final, restrição) {
      // se o vértice inicial e final estão num mesmo submapa
      If (submapa pai do vértice inicial == submapa pai do vértice final) {
            Return buscaCaminho (vértice inicial, vértice final, restrição);
      }

      Int nivelInicial = nível hierárquico do vértice inicial;
      Int nivelFinal = nível hierárquico do vértice final;

      Conjunto_de_caminhos caminhos, caminhos1, caminhos2;

      If (nivelInicial > nivelFinal) {
            caminhos1 = getEsqueletos (vértice inicial, nivelFinal, restrição);
            Caminho aux = novo Caminho só com o vértice final e custo 0;
            caminhos2 = {aux};
      } else if (nivelInicial < nivelFinal) {
            caminhos1 = getEsqueletos (vértice final, nivelInicial, restrição);
            Caminho aux = novo Caminho so com o vértice inicial e custo 0;
            caminhos2 = {aux};
      } else {
            Int nivelComum = primeiroNivelComum (vértice inicial, vértice final);
            caminhos1 = getEsqueletos (vértice inicial, nivelComum, restrição);
            caminhos2 = getEsqueletos (vértice final, nivelComum, restrição);
      }

      caminhos = conectarEsqueletos (caminhos1, caminhos2, restrição);
      return completarMelhorEsqueleto (caminhos, restrição);
}

// Busca o caminho entre dois vértices pertencentes a um mesmo submapa.
// Poderia ser utilizado qualquer método de busca em grafo.
buscarCaminho (vértice origem, vértice destino, restrição) {
      Caminho caminhoDesejado;
      Conjunto_de_Caminhos open, closed;
      Open.add (caminho somente com o vértice inicial);
```

```
        while (open tiver algum elemento) {
              caminho = open.remove(0);
              closed.add (caminho);
              If (vértice final do caminho for igual ao destino) {
                    If (caminhoDesejado é nulo) {
                          caminhoDesejado = caminho;
                    } else {
                          If (custo do caminho < custo do caminhoDesejado) {
                                caminhoDesejado = caminho;
                          }
                    }
              } else {
                    para cada vértice adjacente V do último vértice do caminho {
                          Caminho original;
                          para cada caminho C em closed {
                                If (vértice final do C == V) {
                                      original = c;
                                }
                          }
                          caminho.add(V);
                          if (original é nulo) {
                                open.add(caminho);
                          } else { //se já existe um caminho para aquele vertice
                                If (custo caminho < custo original) {
                                      closed.remove(original);
                                      open.add(caminho);
                                }
                          }
                    }
              }
        }
        Return caminhoDesejado;
}

getEsqueletos (vértice inicial, nível destino, restrição) {

        VerticeSubMapa submapaAtual = submapa pai do vértice inicial;
        Conjunto_de_Caminhos caminhos = estimarCaminhos (vértice inicial, vértices
        pontes do submapaAtual);

        While (nível submapaAtual > nível destino) {
              Conjunto_de_Caminhos caminhosNovos;
              para cada vértice ponte V do submapaAtual {
                    Caminho caminho = novo caminho com custo maximo;
                    para cada caminho C dos caminhos {
```

```
                    Caminho aux = submapaAtual.caminhoPreCalculado (vértice final
do caminho C, vértice V, restrição);
                    Caminho caminhoX = C + aux;
                    If (custo caminhoX < custo caminho) {
                        caminho = caminhoX;
                    }
                }
                caminhosNovos.add (caminho);
            }
            caminhos = caminhosNovos;
            submapaAtual = submapa pai do submapa atual;
        }
        Return caminhos;
}

primeiroNivelComum (vértice inicial, vértice final) {
        Int nivel = nivel hierárquico do vértice inicial;
        VerticeSubmapa v1 = submapa pai do vértice inicial;
        VerticeSubmapa v2 = submapa pai do vertice final;

        While (nivel > 0) {
            nivel --;
            v1 = submapa pai de v1;
            v2 = submapa pai de v2;
            if (v1 == v2) {
                break;
            }
        }
        Return nivel;
}

conectarEsqueletos (caminhos1, caminhos2, restrição) {
        Conjunto_de_Caminhos caminhosRetorno;
        para cada caminho Ci dos caminhos1 {
            Caminho caminhoNovo;
            para cada caminho Cj dos caminhos2 {
                verticeAux = submapa pai do vértice final do caminho Ci;
                Caminho caminhoPre = verticeAux.caminhoPreCalculado (vértice final
do Ci, vértice final do Cj, restrição);
                if (caminhoPre != null) { //caminhos não estao incluídos em
submapas irmãos (caminhoPre eh do tipo 2)
                    caminhoNovo = Ci U caminhoPre U Cj;
                } else {
                    verticeAux = submapa pai do verticeAux;
                    caminhoPre = verticeAux.caminhoPreCalculado (vértice final do
Ci, vértice final do Cj, restrição);
```

```
                        if (caminhoPre != null) { // caminhos estão incluídos em
submapas irmãos (caminhoPre é do tipo 3)
                            caminhoNovo = Ci U caminhoPre U Cj;
                        } else { // nenhum caminho pré calculado disponível
                            caminhoNovo = Ci U Cj;
                            custo caminhoNovo = custo Ci + custo Cj + heuristica
(vértice final Ci, vértice final Cj);
                        }
                    }
                    caminhosRetorno.add (caminhoNovo);
                }
            }
            Return caminhosRetorno;
}

// função que completa o esqueleto do caminho com os vértices faltantes
completarMelhorEsqueleto (caminhos, restricao) {
            Caminho melhorEsqueleto;
            para cada C dos caminhos { // pegar esqueleto de menor custo
                if (custo do caminho < custo do melhorEsqueleto) {
                    melhorEsqueleto = caminho;
                }
            }
            Caminho retorno;
            para cada V dos vértices do melhor esqueleto {
                Caminho temp = buscarCaminho (vértice V, próximo vértice do esqueleto,
restrição);
                If (temp == null) {
                    If (vértice V for do tipo ponte) {
                        temp1 = vértice equivalente ao ponte no nível superior;
                    }
                    If (próximo vértice do esqueleto for do tipo ponte) {
                        temp2 = vértice equivalente ao ponte no nível superior;
                    }
                    temp = buscarCaminho (temp1, temp2, restrição);
                }
                retorno.add (temp);
            }
            Return retorno;
}

estimarCaminhos (vértice inicial, vértices) {
            Conjunto_de_Caminhos caminhos;
            para cada vértice V dos vértices {
                Caminho caminho = vértice inicial + V;
                Custo caminho = heurística (vértice inicial, V);
```

```
            caminhos.add (caminho);
        }
        Return caminhos;
}

// Heurística utilizada: distância "física" entre os dois vértices
Heurística (vértice origem, vértice destino) {
        Return distância entre o vértice origem e o vértice destino;
}


// O pré-calculo do caminho é realizado na inicialização do sistema, após serem
// carregados todos os vértices e arcos da representação do mapa
precalcularCaminhos () {
        para cada submapa SM {
                para cada vértice ponte VP de SM {
                        //Caso 1: caminhos que ligam 2 vértices pontes dentro de um grafo
                        para cada um dos outros vértices pontes VP2 de SM {
                                para cada restrição R possível {
                                        Caminho caminho = buscarCaminho (VP, VP2, R);
                                        SM.addCaminhoPreCalculado (caminho);
                                }
                        }
                        // Caso 2: caminhos que ligam os vértices pontes de um submapa
                        // aos vértices de seu submapa pai
                        para cada vértice ponte VP2 do submapa pai de VP {
                                para cada restrição R possível {
                                        Caminho caminho = buscarCaminho (VP, VP2, R);
                                        SM.addCaminhoPreCalculado (caminho);
                                }
                        }
                        // Caso 3: caminhos que ligam submapas irmãos
                        para cada vértice submapa SM2 do grafo que contem SM {
                                para cada vértice ponte VP2 de SM2 {
                                        para cada restrição R possível {
                                                Caminho caminho = buscarCaminho (VP, VP2, R);
                                                SM.addCaminhoPreCalculado (caminho);
                                        }
                                }
                        }
                }
        }
}
```

REFERÊNCIAS BIBLIOGRÁFICAS

CAGIGAS, D.; ABASCAL, J. Hierarchical Path Search with Partial Materialization of Costs for a Smart Wheelchair. **Journal of Intelligent and Robotic Systems**, v.39, n.4, p.409-431, 2004.

CHAMPANDARD, A. J. Path Planning from Start to Finish. <Disponível em: http://ai-depot.com/BotNavigation/Path.html>. Acesso em: 20 fev. 2005.

DUDEK, G.; JENKIN, M. **Computational Principles of Mobile Robotics**. New York: Cambridge University Press, 2000. Cap.5, p.121-148: Representing and Reasoning about Space.

EKAHAU. Ekahau Positioning Engine – User Guide. Disponível em: <http://www.ekahau.com>. Acesso em: 31 mar. 2005.

ELNAHRAWY, E.; LI, X.; MARTIN, R. Using Area-based Presentations and Metrics for Localization Systems in Wireless LANs. In: *Proceedings of The LCN's Fourth International IEEE Workshop on Wireless Local Networks (WLN 2004)*, Tampa, USA. p.650-657, 2004.

GWON, Y.; JAIN, R.; KAWAHARA, T. Robust Indoor Location Estimation of Stationary and Mobile Users. In: *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '04)*. Hong Kong, China, Mar. 2004.

HAEBERLEN, A.; FLANNERY, E.; LADD, A. M.; RUDYS, A.; WALLACH, D. S.; KAVRAKI, L. E. Practical Robust Localization over Large-Scale 802.11 Wireless Networks. In: *Proceedings of the Tenth ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Philadelphia, USA, 2004. p.70-84.

KUFFNER, J. J.; LAVALLE, S. M. RRT-connect: An efficient approach to single-query path planning. In: *Proc. IEEE Int'l Conf. on Robotics and Automation*, p.995-1001, 2000.

LADD, A. M.; BEKRIS, K. E.; MARCEAU, G.; RUDYS, A.; WALLACH, D. S.; KAVRAKI, L. E. Using wireless Ethernet for localization. In: *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, Sept. 2002. v.1, p.402-408.

LOPES, G; POLICARPO, F; WATT, A. Real-Time BSP-Based Path Planning with A*. In: *Proc. of the 1st Brazilian Workshop in Games and Digital Entertainment*. Fortaleza, Brazil, Oct. 2002.

PLACE LAB. Documentação de API Java open source para localização via wireless. Disponível em: <http://www.placelab.org/>. Acesso em: 21 fev. 2005.

MAHMOUD, Q. Distributed Programming with Java. Manning, 2000.

St LAURENT, S.; JOHNSTON, J.; DUMBILL, E. Programming Web Services with XML-RPC. O'Reilly, 2001. Cap. 3: Client-Server Communication: XML-RPC in Java. Disponível em: <http://www.oreilly.com/catalog/progxmlrpc/chapter/ch03.html>. Acesso em: 10 mai. 2005.

SUN MICROSYSTEMS. Java Remote Method Invocation Specification. Disponível em: <http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html>. Acesso em: 17 mai. 2005.

SUN MICROSYSTEMS. jGuru: Remote Method Invocation (RMI). Disponível em: <http://java.sun.com/developer/onlineTraining/rmi/RMI.html>. Acesso em: 17 mai. 2005.

THAPA, K.; CASE, S. An Indoor Positioning Service for Bluetooth Ad Hoc Networks. In: *Proceedings of the Midwest Instruction and Computing Symposium Conference (MICS 2003)*. Duluth, MN, Apr. 2003.

XIANG, Z.; SONG, S; CHEN, J.; WANG, H.; HUANG, J.; GAO, X. A Wireless LAN-based indoor positioning technology. **IBM Journal of Research and Development,** v.48, n.5/6, p.617–629, 2004. Disponível em: <http://www.research.ibm.com/journal/rd/485/xiang.html>. Acesso em: 28 fev. 2005.