UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Denis Nazareno Hauffe
Fernando Laudares Camargos

SISTEMA PARA GERENCIAMENTO E ADMINISTRAÇÃO DE COMPETIÇÕES DE *TRIATHLON*

Florianópolis, junho de 2004.

Denis Nazareno Hauffe
Fernando Laudares Camargos

SISTEMA PARA GERENCIAMENTO E ADMINISTRAÇÃO DE COMPETIÇÕES DE *TRIATHLON*

Trabalho de Conclusão no Curso de Bacharelado em Ciências da Computação, sob orientação dos professores Dr. João Bosco Mangueira Sobral e Dranda. Daniela Barreiro Claro.

Florianópolis, junho de 2004.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O *triathlon* é um esporte criado no final da década de 1970 e composto de três esportes olímpicos: a natação, o ciclismo e o atletismo (mais precisamente o pedestrianismo). Assim como os esportes que o compõe, o *triathlon* é também um esporte dito *de rendimento* – em toda competição de *triathlon* está presente um cronômetro, utilizado para mensurar a performance dos atletas participantes, controlando, assim, a ordem de chegada dos mesmos.

Este trabalho versa sobre a criação de um novo *sistema de gerenciamento e administração de competições de triathlon*, sendo a base do sistema um cronômetro e uma lista contendo os participantes da competição e suas características individuais.

1.1 O Triathlon

O *triathlon* surgiu como resultado de uma discussão travada entre militares americanos, sediados no arquipélago do Havaí, sobre qual seria o atleta mais resistente: o nadador, o ciclista ou o corredor. Um dos oficiais sugeriu que o atleta mais resistente seria aquele capaz de "competir" nas três provas mais "duras" de cada uma das respectivas modalidades em um mesmo dia. Isso significava "completar" uma travessia marítima de 3,8 quilômetros, depois pedalar os 180 km da "Volta do Havaí" e finalizar com a maratona municipal de Honolulu, de 42,2 quilômetros. Assim foi criada uma das competições de maior prestígio em todo o mundo, batizada de *Ironman*: qualquer homem ou mulher capaz de completar esses 226 quilômetros seria proclamado "homem de ferro".

Paralelamente, na região de *San Diego*, Califórnia, outro tipo de "competição" semelhante acontecia com um outro grupo de "oficiais". Durante o inverno, os salva-vidas da região buscavam manter seu condicionamento físico através da prática de outros esportes que não envolvessem o mar, muito frio nessa época do ano. Constantemente esses esportes eram o ciclismo e a corrida. No começo da temporada seus supervisores organizavam uma espécie de "circuito", envolvendo corrida com obstáculos e um trecho de ciclismo, que tinha finalidade de conferir se seus oficiais se encontravam em boa forma física e aptos para o trabalho. Muitos sustentam a tese de que o *triathlon* tenha "nascido" nesse meio.

Independente de sua origem, o *triathlon* foi crescendo e ganhando força com os anos, despertando a atenção e o interesse de novos adeptos e, principalmente, bons patrocinadores. Ele é hoje um esporte olímpico, e ao contrário da maioria dos esportes que buscam seu espaço nas Olimpíadas, o *triathlon* não teve que passar por um período "probatório" – ele estreiou nas Olimpíadas de Sidney, Austrália, em 2000, distribuindo 6 medalhas. Essa talvez seja a maior prova de sua força e consistência em todo o mundo, pois durante anos foi construída uma estrutura capaz de suportá-lo e sustentá-lo mesmo sem que haja o interesse olímpico

A primeira etapa de uma competição de *triathlon* é a natação, que pode ser realizada no mar, em lagos ou mesmo em rios, como pode ser observado na figura 1. Entre a saída do atleta da natação e o começo do ciclismo ocorre um período chamado de transição; como existem duas transições em uma competição de *triathlon*, essa é conhecida como transição 1, ou simplesmente T1, representada pela figura 2. A figura 3 mostra a etapa de ciclismo e a figura 4 ilustra a segunda transição da competição, ou T2, o momento de encostar a bicicleta e calçar os tênis de corrida. A corrida, vista na figura 5, é a terceira e última modalidade de uma competição de *triathlon*, o momento onde a prova é definida.

Figura 1 – A etapa de natação.

Figura 2 – A transição 1 (T1).

Figura 3 – A etapa de ciclismo.

Figura 4 – A transição 2 (T2).

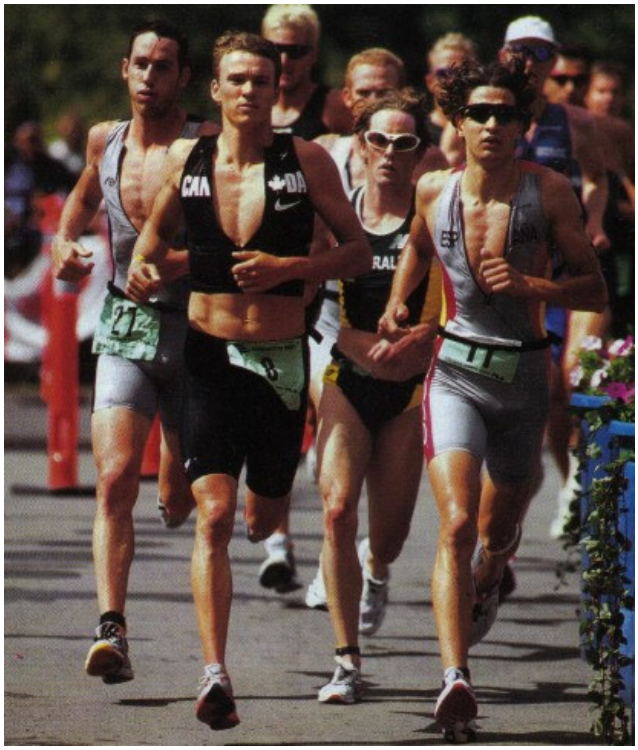

Figura 5 – A etapa de corrida.

Assim como a natação, o ciclismo e o atletismo, o *triathlon* também apresenta múltiplos formatos de competição: em Sidney, assim como deverá ser em todas as próximas Olimpíadas, o formato utilizado foi o Olímpico, constituído de 1500 metros de natação, 40 quilômetros de ciclismo e 10 quilômetros de corrida (1,5km/ 40km/ 10km). No *Ironman* as distâncias são, respectivamente, 3,8 quilômetros, 180 quilômetros e 42,2 quilômetros (3,8km/180km/42,km). Existem vários outros formatos intermediários à estes dois apresentados, ou mesmo que extrapolem esses limites: independente das distâncias, o essencial numa competição de *triathlon* é que a seqüência das modalidades seja sempre a padrão – os atletas nadam, depois pedalam e, por último, correm.

Outra característica interessante do esporte é que ele reúne atletas amadores e profissionais numa mesma prova, muitas vezes, até, numa mesma "largada". Para o atleta amador, na maioria das vezes um estudante ou um trabalhador com um emprego fixo de quarenta horas semanais, que treina de madrugada, durante o horário de almoço e de noite, a oportunidade de "largar" ao lado dos seus ídolos é um momento especial. Mesmo sabendo que ele não conseguirá acompanhar o ritmo deles, o atleta se sente valorizado e motivado a melhorar cada vez mais. Se sente satisfeito. Essa "jogada de *marketing*" dos organizadores dos eventos garante que em uma competição tradicional como o anual *Ironman* do Havaí, dos 1500 atletas participantes ao menos 1400 atletas são amadores – justamente os que pagam os custos da competição e, também, o prêmio dos primeiros colocados profissionais.

Para "acomodar" e beneficiar estes atletas "que fazem a prova acontecer" eles são divididos e classificados em categorias de idade, que reúnem integrantes com diferença de idade máxima de cinco anos (além, é claro, de serem também classificados de acordo com o gênero). Estas categorias de idade possuem um intervalo fixo; assim, um homem de 33 anos vai disputar a categoria "30-34 anos MASCULINO".

1.2 Controle e Gerenciamento de Competições

Atualmente, em uma competição que reúne um grande número de atletas, entre amadores e profissionais, a principal dificuldade encontrada pelos organizadores de eventos é como determinar a posição (colocação) alcançada por um atleta ao cruzar a linha de chegada. Relativo à categoria dos profissionais, a resposta seria óbvia: o primeiro homem e a primeira mulher a cruzarem a linha de chegada serão os campeões. Mas quando a questão é transferida para o âmbito da categoria dos amadores, como, por exemplo, quem é a campeã da categoria '45-50 anos', a perspectiva é outra. Muitas vezes, a campeã desta categoria chega muitos minutos, ou mesmo horas, atrás dos primeiros colocados gerais da competição, e cruza a linha de chegada sem saber que venceu, e, conseqüentemente, sem ter a oportunidade de comemorar sua vitória.

Existem hoje, tanto a nível nacional como a nível mundial, dois sistemas diferentes utilizados para solucionar esse problema do controle das parciais e, conseqüentemente, (da ordem de chegada) dos atletas. Um deles é automatizado e utiliza um *microchip* codificado como controle e o outro é inteiramente manual.

1.2.1 *Microchip* de Controle

A solução encontrada pelos organizadores de provas com maior infra-estrutura foi a adoção de *microchips* de controle contendo os dados individuais do atleta, que o mesmo o carrega durante toda a competição. Ao cruzar a linha de chegada, o *microchip* é identificado por um sensor especial ali colocado, divulgando "instantaneamente" o nome, a categoria e a colocação do atleta num painel eletrônico. Nos outros postos de controle (incluindo aqui as

duas transições) são também colocados sensores, obtendo, dessa maneira, os tempos parciais de natação, da primeira transição, de ciclismo, da segunda transição e da corrida.

Figura 6 - O *microchip* codificado e embalado.

Figura 7 - *Microchip* preso ao calcanhar.

Essa tecnologia do *microchip* de controle é oferecida por diversas companhias e adotada pelas três maiores empresas organizadoras de competições de *triathlon* no Brasil. A *N. A. Promoções*, responsável pela organização do *Triathlon Internacional de Santos* (única competição brasileira à somar pontos para o *Triathlon Pro Tour*, que teve a participação de 613 atletas na sua edição de 2003) e do *Troféu Brasil de Triathlon* (composto por 5 etapas, com uma média de 500 participantes por etapa) utiliza equipamentos da *Chronomix Corp*. A *CIA de Eventos*, responsável pelo *Long Distance Triathlon*, realizado em Pirassununga, SP, registrou a participação de 317 atletas na sua edição de 2002, e utilizou os *chips* da *Chiptiming* para o controle e gerenciamento da competição. A *Latin Sports S/A,* empresa responsável pela organização da etapa brasileira do *Ironman*, o *Ironman Brasil Telecom*, realizado em Florianópolis (678 participantes), utilizou em 2003 equipamentos da *Championchip*, que também fornece a estrutura necessária para o gerenciamento e controle do *Ironman do Havaí* e da *Maratona de Nova Iorque*.

A principal vantagem da tecnologia que utiliza o *chip* é sem dúvida a divulgação "instantânea" dos resultados. Não é preciso aguardar para saber qual é a colocação alcançada pelo atleta ao cruzar a linha de chegada. Outras duas vantagens do sistema é que ele é estável

e seguro (desde que tenha sido corretamente “instalado” e calibrado) e garante a obtenção de todas as parciais.

A principal desvantagem do sistema é o seu alto custo. Dois dos três organizadores das principais competições de *triathlon* do país que utilizam essa forma de controle das parciais estão em busca de um outro sistema, que seja eficiente mas que apresente um custo menor.

Figura 8 - Posto de checagem com dois sensores, para garantir a identificação do chip.

1.2.2 Controle Manual com Pulseira

Como o aluguel ou a aquisição de toda essa infra-estrutura não cabe no orçamento das organizações de competições de menor porte, na maioria destas os resultados são “pinçados” em planilhas manuais, e a divulgação normalmente acontece muito tempo depois do término da competição.

O controle das parciais é feito através da utilização de uma pulseira de velcro onde ficam “grudadas” três senhas contendo o número do atleta. Este por sua vez deve utilizar a pulseira durante toda a competição e entregar uma dessas senhas ao fiscal de cada posto de controle. O fiscal, então, anota o número da senha e a “hora” (o tempo de prova) em que a senha foi entregue numa planilha de papel. No final da competição, é feita uma ordenação dos

participantes de acordo com a ordem de chegada e a categoria à qual pertencem. São também conferidos os tempos de entrega de cada uma das senhas para certificar que o atleta passou por todos os postos de controle.

A entrega de cada senha é de responsabilidade do atleta e constitui, de uma certa forma, um motivo de preocupação adicional para o mesmo. Essa preocupação desnecessária é uma das desvantagens do sistema. Porém, a principal desvantagem desse sistema é o tempo de latência entre a chegada do atleta e a divulgação dos resultados (colocações), que varia de algumas dezenas de minutos a horas, dependendo do número de participantes.

O único aspecto que justifica a utilização desse tipo de controle, e que não deve ser considerado uma "vantagem", é o seu baixo custo de implementação: tal sistema requer um mínimo de recurso, tais como papel, caneta e relógio.

1.2.3 TriathlonApp

Como explicado anteriormente, existem diferenças significativas entre os dois modelos atualmente adotados no gerenciamento de competições de *triathlon*, sendo o orçamento disponível para o mecanismo de controle que define qual sistema será utilizado.

O novo sistema desenvolvido e aqui apresentado, chamado *TriathlonApp*, absorveu o que há de melhor nos sistemas existentes, resultando num sistema de baixo custo capaz de divulgar os resultados de forma rápida e eficiente.

1.3 Estrutura do Trabalho

A estrutura do trabalho, em relação aos capítulos seguintes, foi organizada da seguinte maneira.

O capítulo dois introduz a tecnologia *Java 2 Micro Edition* (J2ME), a plataforma Java para dispositivos móveis, que constitui a principal tecnologia utilizada na construção do novo sistema.

O novo sistema proposto é apresentado no capítulo três.

A implementação do novo sistema é discutida no capítulo quatro e as conclusões resultantes deste trabalho são apresentadas no capítulo cinco.

2 J2ME, A PLATAFORMA JAVA PARA PEQUENOS DISPOSITIVOS

Este capítulo apresenta uma revisão bibliográfica sobre a tecnologia *Java 2 Plataform, Micro Edition* (J2ME), a linguagem para dispositivos com processadores embarcados que mais cresceu nos últimos anos: em 2002 haviam pouco mais de 50 milhões de telefones celulares com tecnologia Java embutida; em 2003, somente a fabricante Nokia vendeu mais de 100 milhões de celulares com Java e a estimativa é que por volta de 2007 praticamente 100% dos telefones celulares executem Java (Almeida, 2004).

O capítulo começa com uma introdução da tecnologia, contextualizando-a dentro da família Java, para então apresentar mais detalhadamente suas características específicas, enfatizando aquelas utilizadas nesse projeto.

2.1 Introdução ao J2ME

Java 2 Plataform, Micro Edition (J2ME) é a versão da plataforma Java para pequenos dispositivos que contém microprocessadores embutidos (ou embarcados), tais como *Personal Digital Assitents* (PDAs), telefones celulares e conversores de TVs à cabo. Ela é formada por um sub-conjunto das *Application Programming Interface* (API) da versão de Java para computadores desktop e servidores (*Java 2 Platform, Standard Edition* (J2SE)), mantendo a compatibilidade com o J2SE sempre que possível.

As características técnicas da arquitetura de cada um destes dispositivos inviabiliza a compatibilidade total entre as duas versões da plataforma Java. Além disso, a justificativa de

uma versão especifica para pequenos dispositivos ocorre devido ao fato de que tais dispositivos são caracterizados por possuir um baixo poder de processamento, uma pequena quantidade de memória disponível, presença ou ausência de algum nível de conectividade e um *display* com limitações de tamanho e definição.

Analisando essas características em diferentes dispositivos com microprocessadores embutidos é possível observar que, apesar de compartilharem as mesmas características restritivas apresentadas acima, existe uma diferença perceptível entre as arquiteturas de diferentes dispositivos: enquanto um típico aparelho de telefonia celular apresenta um display com uma resolução total de 12.288 pixels (96x128), um PDA possui um display de pelo menos 20.000 pixels (Muchow, 2001). Apesar de não ser uma diferença tão acentuada quando comparados ao display de um desktop convencional (786.432 pixels, 1024x768), ela é significativa em se tratando de construir uma interface para aplicações em pequenos dispositivos.

Em virtude dessas diferenças que existem também entre os dispositivos pequenos, os projetistas da *Sun Microsystems*, criadora da plataforma Java, representada pela Figura 9, concluíram que não seria possível criar um único produto de *software* capaz de atender a essa variada gama de dispositivos. Assim, J2ME foi definida como uma coleção de especificações que definem um conjunto de plataformas, sendo cada um delas adequada para um sub-conjunto dos dispositivos destinados aos consumidores que se enquadram nesse escopo (Topley, 2002).

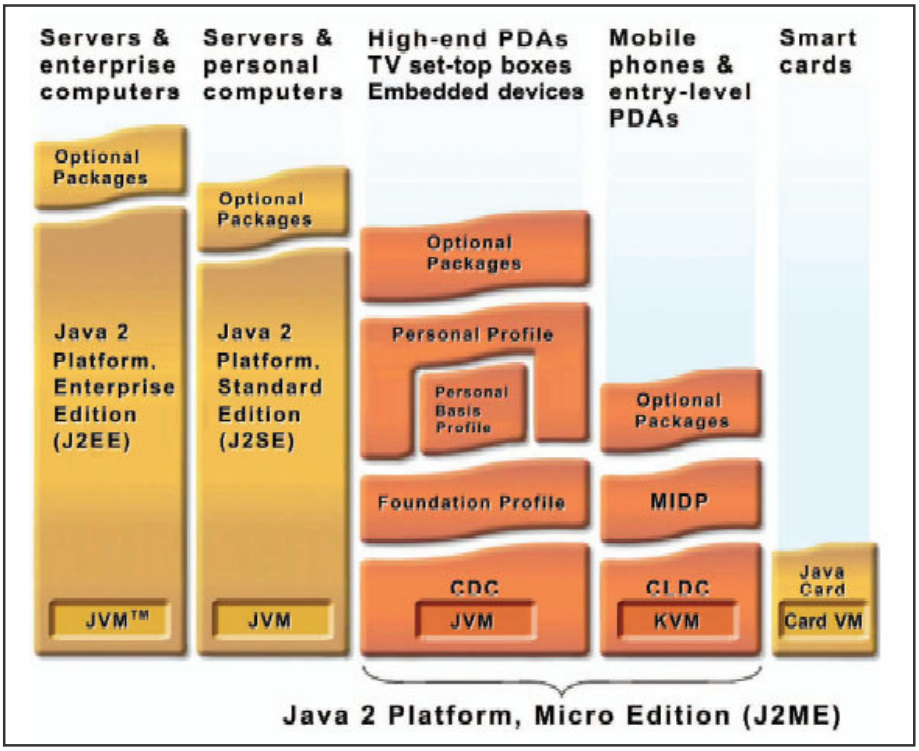

Figura 9 - A plataforma J2ME.

Assim, o sub-conjunto do ambiente completo de programação Java para um dado dispositivo é definido por um ou mais *perfis*, os quais estendem as capacidades básicas de uma *configuração*. A configuração e o perfil (ou perfis) apropriados para um dispositivo dependem tanto das características da sua arquitetura (*hardware*) quanto do mercado ao qual são destinados (Topley, 2002).

A *configuração* esta fortemente ligada à Máquina Virtual Java (*Java Virtual Machine,* JVM), e define os aspectos da linguagem Java e as bibliotecas utilizadas pela JVM para essa configuração particular em razão, principalmente, da disponibilidade de memória, da capacidade do *display*, do grau de conectividade e do poder de processamento (Muchow, 2001).

O *perfil* é visto como uma extensão da *configuração*, e disponibiliza bibliotecas de classes para programação em um dispositivo específico. Foi a forma encontrada para tornar a

J2ME ainda mais específica para um dado dispositivo ao mesmo tempo em que proporciona a ela uma maior flexibilidade aos avanços da tecnologia. O papel do *perfil* é definir APIs para os componentes de interface com o usuário, persistência de informações, manipulação de eventos, *timers*, entre outros, específicos para um ou um número menor de dispositivos que utilizam uma (já específica) *configuração*. A figura 10 ilustra a arquitetura genérica de um dispositivo capaz de executar aplicativos escritos em J2ME: começa pelo Sistema Operacional, como base, seguido pela Máquina Virtual Java (dependente da Configuração), pela Configuração e pelo Perfil.

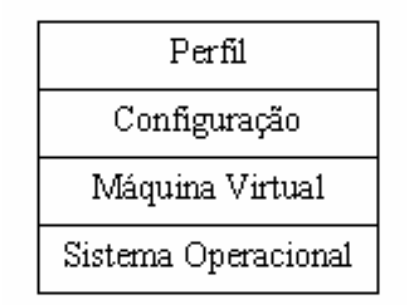

Figura 10 - Arquitetura J2ME genérica.

2.2 A Configuração CLDC

Como visto anteriormente, a *configuração* deve representar a plataforma mínima para o dispositivo alvo. Atualmente, J2ME apresenta duas configurações diferentes: *Connected Limited Device Configuration* (CLDC) e *Connected Device Configuration* (CDC). O que difere uma da outra são os requisitos mínimos que cada uma delas exige. Como o próprio nome já deixa claro, a CLDC, menos exigente, requer um mínimo de 128 Kilobytes de memória para processar Java e 32 kilobytes para alocação de memória em tempo de execução,

proporcionando uma interface com o usuário restrita, e apresentando baixo consumo de energia (geralmente proveniente de baterias) e conectividade tipicamente sem fio, com acesso intermitente e baixa largura de banda. A CDC é uma implementação um pouco mais robusta, exigindo quatro vezes mais memória para processar Java e oito vezes mais memória para ser alocada em tempo de execução (Muchow, 2001).

A maioria dos PDAs e telefones celulares se enquadram entre os dispositivos que utilizam a CLDC, sendo por esta razão a configuração de J2ME utilizada neste trabalho.

2.3 O Perfil MIDP

O perfil complementa a configuração adicionando bibliotecas de classes que proporcionam características apropriadas a um tipo particular de dispositivo. Foi utilizado nesse trabalho o *Mobile Information Device Profile* (MIDP) por ser altamente associado com a maioria dos telefones celulares presentes no mercado, sendo também o único perfil a ter uma implementação compatível com o sistema operacional PalmOS, presente na maior parte dos PDAs de baixo custo. Tal perfil adiciona componentes de interface com o usuário, acesso à rede através do protocolo HTTP 1.1 e persistência de dados à configuração CLDC.

2.3.1 O MIDlet

Os aplicativos Java que funcionam sob dispositivos com o perfil MIDP são conhecidos como MIDlets. Um MIDlet consiste em pelo menos uma classe Java derivada da classe abstrata javax.microedition.midlet.MIDlet.

Os MIDlets (Topley, 2002) são executados em um ambiente de execução dentro da máquina virtual que proporciona um ciclo de vida bem definido através de métodos da classe MIDlet que cada MIDlet deve obrigatoriamente implementar (ainda que não resultem em nada).

Para manter a portabilidade da aplicação, devem ser usadas apenas APIs definidas pela especificação MIDlet.

2.3.2 Máquina Virtual K (KVM)

Muchow (2001) define a Máquina Virtual Java como "o motor por trás da aplicação". Uma vez que o arquivo Java tenha sido compilado (de acordo com a *configuração* em uso) em um arquivo .class e, opcionalmente, incluído em um Arquivo Java (JAR), a JVM traduz o "*byte code*" na linguagem de máquina da plataforma que hospeda a JVM. Ela é ainda responsável pelas questões de segurança, alocação e liberação de memória e manutenção de *threads* em execução.

A máquina virtual para a configuração CDC possui a mesma especificação da máquina virtual presente na J2SE. Para a configuração CLDC, porém, a *Sun Microsystems* desenvolveu uma implementação de referência conhecida como *K Virtual Machine* (KVM), que ocupa entre 40 e 80 kylobytes de memória no dispositivo, além de exigir algo entre 20 e 40 kilobytes de memória dinâmica. O *K* da KVM vem justamente do fato da memória ser medida em Kilobytes, uma forma de chamar atenção para seu pequeno tamanho. O mínimo de processamento que ela exige pode ser fornecido por um processador de 16 bits rodando a 25 Mhz (Muchow 2001).

Apesar de não ser a única implementação da máquina virtual disponível (a IBM, por exemplo, desenvolveu uma versão própria, chamada de *J9*), a KVM da *Sun* foi utilizada no

desenvolvimento desse trabalho, e em conjunto com a CLDC e o MIDP, compõe a "arquitetura específica" de J2ME para os dispositivos alvo utilizados nesse trabalho, conforme ilustra a figura 11 (observando que o sistema operacional para os telefones celulares difere daquele presente nos PDAs).

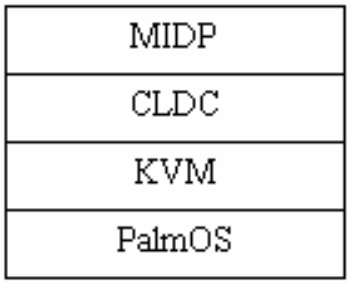

Figura 11 - Arquitetura J2ME específica para os PDAs utilizados.

2.3.3 Persistência de Dados

O armazenamento de informações em pequenos dispositivos como aparelhos de telefonia celular não pode obedecer ao mesmo sistema utilizado em um computador *desktop*, pois tais dispositivos não contam com o apoio de um disco rígido ou mesmo unidades de CD-RW ou de disquete. Além disso, a questão do espaço ocupado pelos dados, a performance do aparelho no acesso e modificação dos registros e as diferentes implementações do sistema de arquivos (quando este existe na arquitetura do dispositivo) desenvolvida pelos diferentes fabricantes torna a persistência de dados um fator crucial a ser suportado por qualquer linguagem de programação.

A linguagem J2ME trabalha essa questão através de uma API (um pacote de classes) pertencente ao perfil MIDP, chamada *Record Management System* (RMS). O RMS pode ser visto como um ambiente de persistência de dados alternativo ao uso de um sistema de arquivos e utiliza memória não-volátil para armazenar os dados.

Muchow (2001) considera o sistema RMS como “uma base de dados orientada a registros” e “que pode ser vista como uma série de colunas em uma tabela com um único identificador para cada coluna”. Assim, cada registro consiste em um *record ID* (um identificador numérico para o registro que atua como a chave-primária da base de dados) e um *array* de *bytes*, onde fica armazenada a informação propriamente dita (independente de qual seja o dado que se queira armazenar, ele deve ser convertido para o formato *byte* antes de ser armazenado, e reconvertido novamente para o seu formato original após ter sido recuperado da base de dados). Uma coleção desses registros (*record*) é chamada de *record store*. “Um MIDlet pode ter quantos *record store* queira (inclusive nenhum), onde cada um deles é identificado através do seu nome” (Muchow 2001). A figura 12, retirada da obra de Martins (2004), representa a estrutura de funcionamento da API RMS.

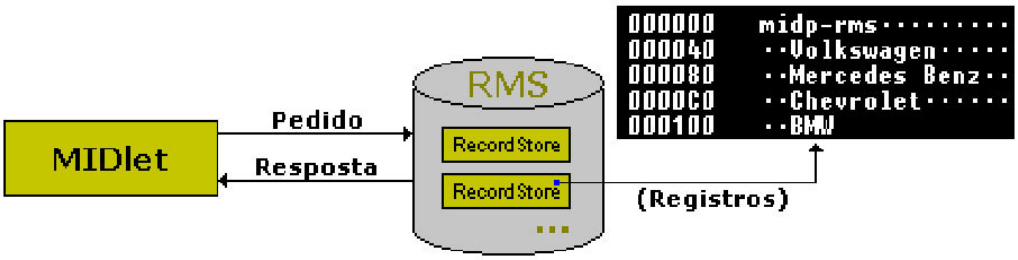

Figura 12: Persistência de dados através da API RMS.

3 MODELO DO SISTEMA PROPOSTO

O sistema TriathlonApp desenvolvido utiliza a tecnologia de aparelhos de telefonia celular e/ou de PDAs (*Portable Digital Assistent*, visto na figura 13) em substituição às planilhas de papel para a obtenção das parciais (um dispositivo para cada parcial a ser capturada e um adicional (opcional) para controlar a ordem de chegada dos atletas e anunciar a colocação de cada um deles), e um computador servidor, que hospeda a base de dados do sistema e fornece os serviços relacionados à manutenção dessa base de dados.

A conexão entre os dispositivos que realizam o controle das parciais e o servidor agilizará de forma considerável a divulgação da lista com os resultados finais e parciais, além de fornecer "instantaneamente" a colocação geral e por categoria de cada atleta que cruzar a linha de chegada.

Figura 13 - PDA (Portable Digital Assistant).

O número de dispositivos de controle a ser utilizado varia de acordo com a quantidade de parciais que se deseja "capturar". Ou seja, se o tempo referente a cada uma das duas transições (T1 e T2) for "capturado" separadamente, serão necessários um mínimo de 5 dispositivos (um para o tempo de natação, um para o tempo da T1, um para o tempo de

ciclismo, um para o tempo da T2 e um para o tempo de corrida), sendo um sexto dispositivo utilizado, opcionalmente, como auxiliar na identificação isolada de cada atleta pela organização da prova em situações diversas (como, por exemplo, para que o locutor da competição também possa identificar os atletas que cruzam a linha de chegada). Para reduzir ainda mais o custo do sistema, é possível trabalhar com somente 3 dispositivos, sendo o tempo total de prova obtido à partir do dispositivo que controla a parcial da corrida, última etapa do *triathlon.* Em tal situação, o tempo referente às transições será "embutido" em uma das parciais (geralmente, sobre a parcial do ciclismo).

Se, por algum motivo, não houver interesse em capturar as parciais intermediárias é possível ainda trabalhar com um aparelho apenas, coletando os tempos finais na linha de chegada ao mesmo tempo em que é anunciada a colocação de cada participante.

Um relatório final de prova contendo os tempos "puros" das três modalidades, sem que o tempo de "latência" das transições seja embutido em uma delas, é particularmente importante para o atleta e seu treinador, que os utilizam como "comparativo" na avaliação da sua performance. Assim, por exemplo, o tempo relativo ao ciclismo será realmente o tempo que o atleta gastou para cobrir o percurso de ciclismo, e não o tempo gasto no ciclismo mais o tempo gasto nas transições.

3.1 Esquema de Funcionamento do Novo Modelo: uma abordagem prática

Como mencionado anteriormente, podem ser utilizados de 1 a 5 dispositivos para registrar o controle da competição, dependendo das parciais que se queira obter. Independente da opção, o modelo proposto fará a automatização dos resultados obtidos manualmente, com

base em um banco de dados, previamente em produção. A inserção de dados nesse banco, ou seja, a lista com a relação dos atletas participantes, deve ser feita antecipadamente para que, no momento da competição, cada atleta possa ser identificado através de um número que o identifica na lista de registros (o mesmo número contido na pulseira de identificação e impresso num pedaço de papel em formato retangular que o atleta deve carregar no peito e/ou nas costas durante as etapas de ciclismo e corrida).

3.1.1 O Banco de Dados

A inscrição para uma competição de *triathlon* é feita com antecedência e dificilmente (ou praticamente nunca) são aceitas inscrições "em cima da hora". Isso é feito para que os organizadores e promotores do evento possam preparar a competição de acordo com a quantidade de atletas participantes, classificando-os nas suas respectivas categorias e providenciando a estrutura necessária para garantir a realização da prova.

Quando um atleta se registra, os principais dados que ele deve fornecer são seu nome completo, gênero, data de nascimento e a definição da sua participação como atleta amador ou como atleta profissional (categoria também conhecida como "elite").

A data de nascimento, para os atletas amadores, é o dado que os classificará dentro de uma lista pré-definida (e geralmente padrão) de categorias por idade. A regra que dita a categoria do atleta, pode variar de acordo com a organização da prova: o campo "idade do atleta", a ser utilizado como elemento de classificação do mesmo nas categorias de idade, pode tanto considerar a própria idade do atleta no dia prova, como a idade do atleta no primeiro dia do ano, ou mesmo a idade do atleta no último dia do ano. Assim, no ato da inscrição, o atleta amador não determina a faixa de idade na qual vai competir; o próprio sistema (ou melhor, a organização da competição) é que vai determinar isso, com base na data

de nascimento do atleta e na regra utilizada para determinar a faixa de idade em que o mesmo se enquadra.

Uma das maneiras usuais de realizar essa classificação é apresentada pela figura 14. O critério de classificação prevê ainda a classificação de acordo com o gênero, tanto para profissionais quanto para amadores.

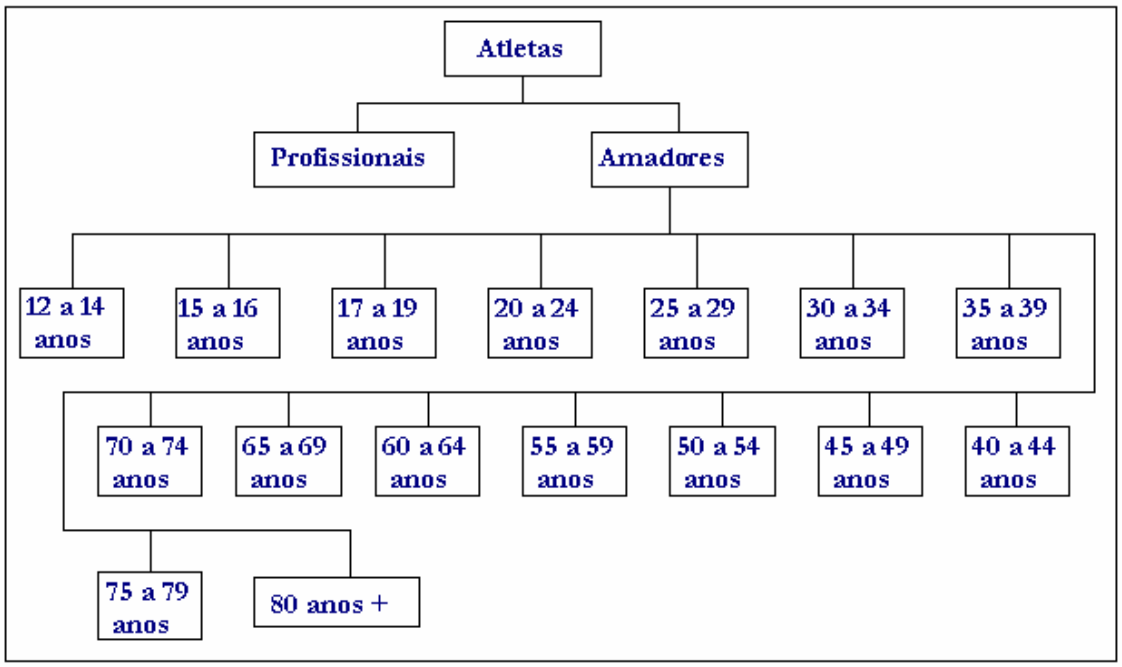

Figura 14 - Classificação dos atletas.

No *triathlon*, como em todo tipo de competição que abrange um número grande de participantes, é usual que cada atleta seja identificado por um número. Após ter sido registrada a sua inscrição, cada atleta recebe um número de identificação. O mesmo deve competir com esse número fixado em seu corpo no dia da prova, de forma que a organização possa identificá-lo facilmente. Esse número é a peça chave em todo sistema de controle e gerenciamento de competições e é através dele que o atleta é identificado no banco de dados do sistema.

O registro presente no sistema antes do início da competição é uma lista de participantes, onde cada atleta ocupa uma linha. Cada uma dessas linhas é composta pelo seu número de identificação, nome e a categoria a qual pertence (um quarto item seria o patrocinador, clube ou associação ao qual ele pertence, mas isso não pertence ao escopo deste trabalho). A tabela 1 apresenta uma lista como esta (fictícia) que ilustra melhor a situação.

Nº id	Nome	Categoria
1	Pedro	M Elite
20	João	M 20-24
52	Paula	F 35-39
74	Lúcia	F 40-44
41	Luis	M 30-34
45	Marcelo	M 30-34
13	Davi	M 17-19
2	Ivan	M Elite
21	Ricardo	M 20-24
3	Letícia	F Elite
4	Lucas	M Elite
54	Fernanda	F 35-39
22	Alexandre	M 20-24
43	Renato	M 30-34
72	Raquel	F 40-44
5	Peter	M Elite
6	Milton	M Elite
55	Priscila	F 35-39
7	Daniela	F Elite
23	Rodrigo	M 20-24
44	Antonio	M 30-34
24	Mauro	M 20-24
8	Felipe	M Elite
25	Mauro	M 20-24
58	Natalia	F 35-39

Tabela 1 - Lista de atletas.

A lista com os resultados finais, a ser divulgada no encerramento da prova, é a própria lista de participantes acrescentada de informações adicionais. Os itens adicionados são: colocação geral, colocação na categoria, tempo total de prova e os tempos parciais (estes dependendo do número de dispositivos utilizados no controle e registro das parciais, sendo que, no caso ideal, deverão ser registrados, além do tempo total de prova, o tempo de natação, da primeira transição, do ciclismo, da segunda transição e da corrida).

A lista com os resultados finais costuma ser ordenada de acordo com a classificação geral (CG), ou seja, do menor, para o maior tempo de prova. Listas complementares podem ser fornecidas à parte, divulgando, por exemplo, os resultados de cada categoria separadamente – colocação na categoria (CC).

A Tabela 2 apresenta uma lista (também fictícia) com os resultados finais de uma competição.

CG	CC	Nº	Nome	Categoria	Natação	T1	Ciclismo	T2	Corrida	Total
1	1	2	Ivan	M Elite	00:27:50	00:00:55	02:14:50	00:00:38	01:16:00	04:00:13
2	2	5	Peter	M Elite	00:26:55	00:00:47	02:15:24	00:00:34	01:19:27	04:03:07
3	3	9	Eduardo	M Elite	00:28:19	00:00:49	02:19:51	00:00:31	01:18:08	04:07:38
4	4	1	Pedro	M Elite	00:28:22	00:00:57	02:16:57	00:00:42	01:20:42	04:07:40
5	5	4	Lucas	M Elite	00:28:24	00:00:58	02:21:23	00:00:31	01:26:39	04:16:26
6	6	6	Milton	M Elite	00:33:07	00:00:59	02:27:26	00:00:38	01:19:18	04:21:28
7	1	13	Davi	M 17-19	00:30:26	00:01:01	02:27:07	00:00:37	01:26:09	04:23:42
8	7	8	Felipe	M Elite	00:27:58	00:01:03	02:30:46	00:00:42	01:25:06	04:25:35
9	1	21	Ricardo	M 20-24	00:35:56	00:00:58	02:25:40	00:00:47	01:24:24	04:27:45
10	2	22	Alexandre	M 20-24	00:32:08	00:01:13	02:23:44	00:00:35	01:31:07	04:28:47
11	1	45	Marcelo	M 30-34	00:31:29	00:01:04	02:28:15	00:00:48	01:27:45	04:29:21
12	1	7	Daniela	F Elite	00:33:02	00:01:05	02:31:09	00:00:39	01:23:46	04:29:41
13	3	24	Mauro	M 20-24	00:35:42	00:00:58	02:26:05	00:00:37	01:26:50	04:30:12
14	2	44	Antonio	M 30-34	00:30:20	00:01:14	02:30:05	00:00:31	01:28:34	04:30:44
15	2	10	Eliza	F Elite	00:34:28	00:01:12	02:26:55	00:00:45	01:28:54	04:32:14
16	3	3	Letícia	F Elite	00:28:24	00:01:03	02:29:15	00:00:48	01:34:59	04:32:38
17	3	41	Luis	M 30-34	00:31:02	00:01:25	02:33:44	00:01:07	01:25:41	04:32:58
18	4	25	Mauro	M 20-24	00:28:15	00:01:18	02:30:20	00:01:09	01:33:31	04:34:33

Tabela 2 - Lista com resultados.

3.1.2 Captura das Parciais

A captura das parciais é feita a partir do número de identificação do atleta. Quando o mesmo passa em um dos postos de controle, o fiscal responsável informa o número do atleta no dispositivo de controle e o sistema registra a parcial.

Para evitar confusão, caso um grande número de atletas passe pelo posto de controle ao mesmo tempo, é indicado o uso de mais de um fiscal em cada posto. Assim, um fica responsável pelo registro e outro, ou outros, fica responsável pela visualização do número de identificação dos atletas.

A sincronização dos cronômetros do sistema de cada dispositivo de controle é um ponto importante e delicado do sistema. Todos os dispositivos devem ter os cronômetros da aplicação inicialisados simultaneamente, no momento da largada. A captura de cada parcial é feita sobre o tempo total de prova e a "filtragem" da parcial (o tempo real correspondente a cada parcial) é feita no final da prova, subtraindo-se o tempo obtido pelo(s) tempo(s) registrado(s) pela(s) parcial(ais) anterior(es).

3.1.3 Exemplo de Funcionamento

A figura ilustra um exemplo de funcionamento do novo sistema em uma competição de *triathlon*. O exemplo ilustra a situação ideal, que conta com 5 dispositivos registrando as parciais e 1 dispositivo auxiliar. Todas as "configurações mais simples" (com menos dispositivos de controle) podem ser abstraídas à partir desta.

Em um primeiro momento, os dispositivos de controle têm seus cronômetros sincronizados pelos respectivos fiscais de prova (eles são simultaneamente iniciados no

momento da largada). Após a “sincronização”, cada fiscal deve se dirigir ao seu Posto de Controle.

Ao completar o percurso de natação, o atleta passa pelo Posto de Controle 1, onde é registrado o seu tempo de natação no PDA1.

Após passar pelo primeiro Posto de Controle, o atleta se dirige ao local onde estão as bicicletas, pega sua bicicleta e parte para a etapa de ciclismo. No momento em que o atleta sai do local onde ficam as bicicletas, seu tempo de prova é registrado pelo PDA2 no segundo Posto de Controle. O tempo gasto pelo atleta na primeira transição da prova (T1) é obtido através da subtração do tempo registrado no PDA2 pelo tempo registrado no PDA1.

Concluindo a etapa de ciclismo, o atleta retorna com a sua bicicleta ao mesmo local de onde partiu com ela. Seu tempo de ciclismo é registrado pelo PDA3 (terceiro Posto de Controle) assim que o atleta adentra essa área (o tempo real gasto na etapa de ciclismo é obtido, posteriormente, pela subtração do tempo registrado no PDA3 pelo tempo registrado no PDA2).

O PDA4 (quarto Posto de Controle) registra o tempo do atleta no momento em que ele começa a etapa de corrida. A diferença entre o tempo registrado no PDA4 e o tempo registrado no PDA3 é igual ao tempo gasto pelo atleta na segunda transição da prova (T2).

Por fim, o tempo total de prova do atleta é computado pelo PDA5, no momento em que o atleta cruza a linha de chegada (quinto Posto de Controle). Nesse PDA fica registrada a ordem de chegada dos participantes (resultado geral). O tempo que o atleta gastou para percorrer a etapa de corrida é encontrado pela subtração do tempo registrado no PDA5 pelo tempo registrado no PDA4.

O PDA6 é utilizado como um PDA de apoio. O fiscal que o opera deve se posicionar próximo ao local de encerramento da prova, de forma a identificar o atleta que está prestes a cruzar a linha de chegada (consultando a lista de atletas participantes no banco de dados) e

comunicando o nome e a classificação “virtual” do atleta (via rádio) para o locutor da prova. Assim, é possível “conferir” se o atleta que está prestes a cruzar a linha de chegada é o primeiro colocado (campeão) da sua categoria de idade.

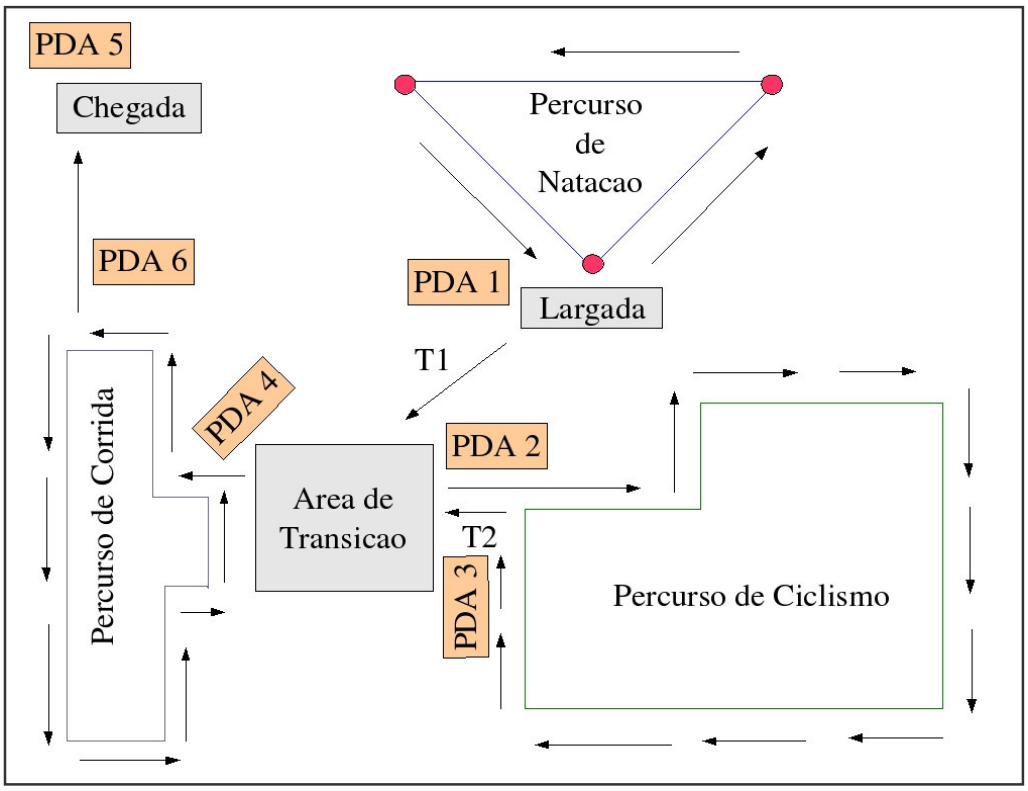

Figura 15 - Exemplo do novo sistema.

3.2 Especificações Técnicas

Como citado anteriormente, um dos atrativos do novo sistema é o seu baixo custo. Porém, quando se considera a utilização de dispositivos eletro-eletrônicos, como os PDAs e aparelhos de telefonia celular, sempre existe um custo inerente.

Hoje em dia existem diversos fabricantes de dispositivos oferecendo uma grande diversidade de modelos. Através de uma simples e rápida pesquisa de mercado é possível descobrir qual modelo de qual fabricante apresenta o menor custo. Porém, como tudo sujeito às regras de mercado, a situação está sujeita a mudanças, e logo pode surgir um outro modelo, de um outro fabricante, sendo vendido por um preço mais atrativo.

Quando chega o momento de escrever um *software* para esse novo sistema, a especificação do mesmo deve ser analisada cuidadosamente. Optar por um determinado modelo de dispositivo, de um determinado fabricante, que ofereça os melhores custo-benefícios (no momento), e torcer para que isso não mude por um longo tempo (ou que o modelo não seja "descontinuado") não é uma solução a ser considerada.

A dificuldade nesse ponto está na existência de uma grande variedade de diferentes arquiteturas de dispositivos, e uma variedade um pouco menor de sistemas operacionais para esses aparelhos de pequeno porte. Para que o novo sistema possa realmente atender ao quesito baixo orçamento, ele não pode ter sua utilização restrita a apenas um ou a um pequeno grupo de aparelhos: a situação ideal é que ele funcione no maior número possível de modelos, de diferentes fabricantes.

3.2.1 Portabilidade e Escolha da linguagem de Programação

Diante da situação apresentada anteriormente, o sistema deve ser portável, de forma a permitir sua utilização em diferentes sistemas operacionais e diferentes arquiteturas. Essa característica direciona a escolha da linguagem de programação a ser utilizada. Uma das possibilidades é utilizar uma linguagem "interpretada" por uma máquina virtual. Assim, desde que a máquina virtual dessa linguagem esteja instalada no dispositivo, o programa deverá ser capaz de executar nele sem maiores dificuldades.

A linguagem deve ainda possibilitar a comunicação e a transferência de dados entre os dispositivos de controle e um servidor.

Diante deste quadro de requisitos, foi escolhida a linguagem Java, mais especificamente a versão para pequenos dispositivos da plataforma, conhecida como J2ME™ (*Java™ 2 Platform, Micro Edition*). J2ME™, é uma derivação de Java™, que é uma linguagem de livre distribuição da *Sun Mycrosystems™*. Ela possibilita a interação com a Internet e possui uma grande disponibilidade de APIs (*Application Programming Interface*) que possibilitam, por exemplo, a implementação de persistência de dados nos dispositivos, o uso do protocolo HTTP de conexão e outros recursos *wireless* (como a tecnologia *bluetooth*), possibilitando futuras modificações e expansões do novo sistema.

4 DESENVOLVIMENTO DO SISTEMA

Esse capítulo descreve o desenvolvimento do sistema proposto, apresentando, inicialmente, uma descrição das ferramentas utilizadas, para então introduzir a implementação do mesmo, dividida em duas etapas. A primeira delas é caracterizada pela parte da aplicação que é executada em um computador servidor: ela inclui o banco de dados onde será armazenada a informação equivalente aos participantes da competição, a interface *web* através da qual esse banco de dados é acessado e também a interface através da qual o aplicativo MIDlet, presente nos dispositivos móveis, acessa as informações presentes no banco de dados. A segunda etapa é caracterizada pela parte da aplicação que é executada no(s) dispositivo(s) móvel(eis) (telefone celular e/ou PDA), no formato de um aplicativo MIDlet. Através desse aplicativo é realizado o controle dos resultados da competição de *triathlon*. Cada uma das duas etapas do sistema será apresentada a seguir.

4.1 Descrição do Ambiente

Foram utilizadas, no desenvolvimento do sistema, as ferramentas *Java 2 Micro Edition Wireless Toolkit* versão 1.0.4_01 (WToolkit), da *Sun Microsystems*, juntamente com o editor *Eclipse Plataform* versão 3.0, da IBM. Tais ferramentas foram utilizadas para a confecção e compilação do código e para a execução do aplicativo.

Aplicativos desenvolvidos para computadores *desktop* exigem, na maioria das vezes, que o código seja compilado à partir do mesmo. Em se tratando de dispositivos com

processadores embarcados, o código também é escrito e compilado à partir de um *desktop*, porém executado no dispositivo. Como o processo de construção do aplicativo exige uma série repetitiva do processo de edição, compilação e execução, até que todos os erros sejam encontrados e corrigidos, os fabricantes dos dispositivos disponibilizam emuladores dos mesmos, *softwares* que possuem as mesmas propriedades de funcionamento dos próprios dispositivos mas que funcionam à partir de um computador *desktop*. Assim, não é necessário transportar o aplicativo para o dispositivo à cada nova mudança no código. Os testes podem ser efetuados no emulador fornecido pelo fabricante.

O WToolkit já é instalado com alguns emuladores. Foram adicionados à lista o *Palm OS Emulator* (POSE), o emulador para dispositivos com sistema operacional PalmOS. O WToolkit foi utilizado tanto de forma independente como na forma de um *plug-in* do Eclipse (incorporando ao mesmo à possibilidade de compilar código J2ME).

O compilador que gera o *bytecode* compatível com o conjunto CLDC+MIDP está presente no WToolkit. Depois que o aplicativo tenha sido exaustivamente testado e se encontre em sua versão definitiva, é criado, através do WToolkit, um Arquivo Java (.JAD) contendo o *bytecode* do aplicativo e um Arquivo '.JAR' que contém as informações inerentes ao mesmo (como versão do aplicativo, perfil utilizado, configuração utilizada, etc). Para transportar o aplicativo para a maioria dos telefones celulares basta publicar esses arquivos em um repositório da Internet e utilizar o mecanismo de *download* presente nos aparelhos para obter uma cópia dos arquivos. Para utilizar o aplicativo num dispositivo com sistema operacional PalmOS é preciso converter esses mesmos arquivos (.JAD e .JAR) para o formato PRC (nativo da linguagem PalmOS). Esse processo pode ser realizado através do *PRC Converter Tool*, um aplicativo da própria *Sun Microsystems* para essa finalidade. Uma vez convertido, o arquivo '.PRC' pode ser copiado do *desktop* para o PDA através de cabo serial/USB.

O aplicativo gerenciador de banco de dados utilizado foi o MySQL, versão 4.0.18, e como servidor *web* foi utilizado o Apache Tomcat 5.0. A versão de Java instalada no servidor, e acessada pelo navegador para compilar os *servlets* gerados pela linguagem JSP, foi o J2SDK 1.4.2_01.

As ferramentas de trabalho citadas foram utilizadas tanto sobre o sistema operacional Microsoft Windows XP Professional, versão 2002, quanto sobre o sistema operacional Linux Mandrake, versão 9.1. O trabalho foi desenvolvido paralelamente em ambas as plataformas.

4.2 Decisão de Projeto: Comunicação entre as Partes

Uma das partes críticas do projeto foi definir como a lista com a relação dos atletas participantes da competição seria montada e teria manutenção, além de poder ser "importada" pelo MIDlet. A submissão manual da relação diretamente no dispositivo representava uma tarefa difícil e pouco prática (devido à falta de um teclado completo nos aparelhos), além de caracterizar um trabalho "em duplicata", pois certamente a organização da prova precisaria ter essa mesma lista disponível por inúmeros outros motivos. Dentre as possibilidades disponíveis, optou-se por utilizar um banco de dados que pudesse ser acessado pela Internet através de uma página HTML: assim, o responsável por inserir as informações nessa base de dados (a lista dos participantes) poderia fazê-lo acessando o servidor contendo o banco de dados de qualquer computador com acesso à Internet.

Para manter o sistema funcional com a utilização de *softwares* de livre distribuição e a interação com a linguagem Java, optamos por um conjunto bastante comum e difundido de aplicações: o banco de dados MySQL e a tecnologia *Java Server Pages* (JSP) aliada ao

Apache Tomcat, o *container* de *servlets* (“aplicações” JSP) usado na implementação de referência da tecnologia JSP.

A interação entre as tecnologias se dá da seguinte forma: inicialmente, o cliente faz uma requisição através de um *browser* de uma página JSP, que nada mais é do que um documento em formato texto que combina *tags* HTML com as *tags* JSP, sendo a página, então, processada pelo servidor. Se for a primeira vez que a página é requisita pelo *browser* ela será transformada em um programa Java (um *servlet*), sendo este compilado, gerando um bytecode que, por sua vez, gera uma página HTML que é enviada de volta ao *browser* do cliente. A partir da segunda vez que esta mesma página for acessada, é verificado se ocorreu alguma modificação na mesma. Se não ocorreu, o *bytecode* armazenado é chamado diretamente, e se ocorreu, o processo de compilação feito pelo *browser* se repete.

Como uma das únicas formas de comunicação dos MIDlets com “o mundo exterior” é através do protocolo HTTP, utilizamos o servidor Apache para hospedar os *servlets* que disponibilizam as informações do banco de dados MySQL, fazendo com que seja possível para o MIDlet acessar (ainda que indiretamente) o banco de dados.

4.3 Primeira Etapa da Implementação: o servidor

A parte central da primeira etapa é o banco de dados contendo a lista dos participantes da competição e suas principais características, que será manipulada pelo sistema. O banco de dados foi implementado através do MySQL. Nele foi criada uma base de dados chamada “triathlon”, que hospeda a tabela principal do banco, que recebeu o nome de “participantes” e é ilustrada pela figura 16.

```
C:\MYSQL\BIN>mysql triathlon
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 22 to server version: 4.0.18-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show fields from participantes;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| numero    | int(11)     |      | PRI | 0       |       |
| nome      | varchar(40) | YES  |     | NULL    |       |
| sexo      | char(1)     | YES  |     | NULL    |       |
| CG        | int(11)     | YES  |     | NULL    |       |
| CC        | int(11)     | YES  |     | NULL    |       |
| tempo     | varchar(10) | YES  |     | NULL    |       |
| categoria | varchar(6)  | YES  |     | NULL    |       |
| catID     | int(2)      | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

Figura 16 - Campos da tabela participantes.

O número do atleta é a chave primária da tabela. Os outros campos que a compõe são o nome do atleta, sexo, categoria, identificador numérico da categoria (catID), colocação geral (CG), colocação na categoria (CC) e tempo final (tempo). Foi feita a opção de registrar apenas o tempo final de prova, e não todas as parciais de cada modalidade. Essa decisão facilitou o processo de modelagem e teste do sistema e não impede a inserção no registro das demais parciais em um momento futuro, sendo necessário apenas incluir outras três colunas (tempo de natação, tempo de ciclismo e tempo de corrida) na tabela participantes e fazer os ajustes necessários nas partes correspondentes do sistema.

O banco de dados é acessado através de uma página *web* JSP. O aplicativo Apache Tomcat foi utilizado para "transformar" o computador *desktop* em um servidor, de modo a disponibilizar uma pasta virtual do diretório raiz (D:\testeJSP) como um endereço da Internet. Assim, todos os arquivos dessa pasta virtual poderiam ser acessados através do endereço http://localhost:8080/testeJSP/, onde o termo 'localhost' significa "neste computador" e o número "8080" representa a porta através da qual acontece a comunicação. Foi criado um subdiretório da pasta 'testeJSP' chamado 'gerencia', onde foram salvos todos os arquivos JSP com exceção do arquivo responsável pelo acesso ao banco de dados, chamado 'acesso.jsp'.

Esse arquivo ficou hospedado em um subdiretório da pasta gerencia, chamado ‘conexao’, e foi utilizado por todos os outros arquivos JSP que precisam se comunicar com o banco de dados.

O registro de atletas no banco de dados, e sua posterior manutenção, foi implementado através de uma página JSP principal, chamada ‘listagem.jsp’. A figura 17 ilustra a mesma.

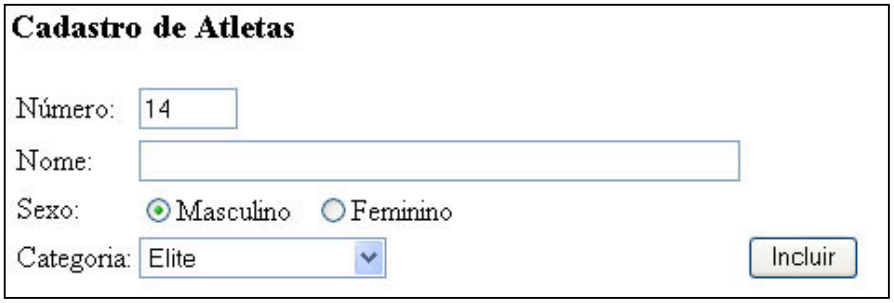

Figura 17 - Cadastro de atletas, página ‘listagem.jsp’.

O número do atleta é fornecido pelo sistema: não é possível escolher um número em especial. Alguns organizadores de competições podem não apreciar esse empecilho, mas como o número do atleta foi utilizado como chave primária no banco de dados, essa alternativa apresenta maior segurança. A pessoa responsável pelo cadastro do atleta deve informar o nome do mesmo, seu sexo e categoria. Ao pressionar o botão ‘incluir’, o sistema registra o atleta e seus dados na tabela participantes do banco de dados.

A medida em que os atletas vão sendo adicionados ao banco de dados, o sistema apresenta a tabela contendo os dados desses atletas e a opção por ‘eliminar’ o atleta do registro ou ‘alterar’ os dados do mesmo. Essa foi a forma escolhida para atualizar as informações no banco de dados. Para reinicializar todos os registros e iniciar um novo cadastro para uma competição diferente, basta excluir todos os atletas do banco. Somente após a exclusão de todos os atletas do banco é que o contador do número dos atletas será

reinicializado, disponibilizando o número um novamente para uso. A figura 18 traz a mesma página de cadastro acompanhada da tabela contendo os participantes já cadastrados.

Figura 18 - Página de cadastro e manutenção do registro de atletas.

Os demais campos da tabela participantes (CG, CC e tempo) são iniciados com valor nulo. Ao final da competição eles serão preenchidos com os respectivos valores para cada atleta.

A página ‘listagem.jsp’ apresenta ainda a possibilidade de gerar uma lista com a relação dos atletas inscritos para impressão ou gerar a lista com os resultados da competição,

caso a mesma já tenha sido realizada e os dados referentes a classificação dos atletas já tenham sido exportados para o banco de dados.

4.4 Segunda Etapa da Implementação: a aplicação do dispositivo móvel

A segunda etapa é caracterizada pelo aplicativo MIDlet, presente no dispositivo móvel, que chamamos de 'TriathlonApp'. Uma vez iniciado, é apresentado o menu principal do aplicativo, com as opções 'Importar dados', 'Cronômetro', 'Exportar Dados', 'Salvar Dados', 'Recuperar Dados', 'Sobre' e 'Sair', como ilustra a figura 19. Cada uma dessas opções será apresentada individualmente a seguir.

Figura 19 - Menu principal do MIDlet emulado em dois dispositivos diferentes.

4.4.1 Importar Dados

Antes que o controle da competição possa ter início, é preciso que o aplicativo "conheça" os atletas participantes. Para tanto, é preciso que o sistema importe a relação contendo a lista dos atletas participantes do banco de dados. Isso é feito através da primeira opção do menu principal do aplicativo "TriathlonApp". Quando essa opção é selecionada, o aplicativo, através do método 'addNumeros()' da classe BaseDeDados, acessa a página JSP 'números.jsp' no endereço http://localhost:8080/testeJSP/gerencia/numeros.jsp. A página faz uma consulta ao banco de dados e retorna o número de cada um dos participantes cadastrados. De posse desses números, o MIDlet acessa, através do método 'addAtletas()', uma segunda página, chamada 'quote.jsp', fornecendo como parâmetro o número do primeiro atleta. A figura 20 ilustra os métodos 'addNumeros()' e 'addAtletas()', da classe BaseDeDados.

Esta segunda página faz uma pesquisa no banco de dados, usando o número do atleta como entrada, e retorna o nome e a categoria do atleta. Esse processo é repetido para cada atleta (ou número). O MIDlet cria um objeto atleta, da classe atleta, para cada atleta registrado no banco de dados com as informações obtidas em cada consulta. Assim, ao final do processo teremos um *array* de objetos da classe 'atletas' contendo as mesmas informações presentes na tabela 'participantes' do banco de dados. É através dessas informações que será possível identificar o nome do atleta e sua categoria quando este cruzar a linha de chegada, determinando o seu tempo final de prova e sua colocação geral e na categoria.

```
private boolean addNumeros()
{
  int i;
  String strNum = null;
  String strInsc = null;
  try
  {
    strInsc = conexao.getsetData("numeros.jsp");
  }
  catch(Exception e)
  {
    e.printStackTrace(); //Para fins de debug
  }
  i = strInsc.indexOf('*');
  strInsc = strInsc.substring(i+1);
  while(strInsc.length() > 1)
  {
    i = strInsc.indexOf(' ');
    strNum = strInsc.substring(0, i);
    numeros.addElement(strNum);
    strInsc = strInsc.substring(i+1);
    //System.out.println(strNum); //Para fins de debug
  }
  return true;
}


private boolean addAtletas()
{
  String strData = null;
  for(int i=0; i<numeros.size(); i++)
  {
    try
    {
      strData = conexao.getsetData("quote.jsp?num=" + (String)numeros.elementAt(i));
    }
    catch(Exception e)
    {
      e.printStackTrace(); //Para fins de debug
    }
    String nome = strData.substring(strData.indexOf('*')+1, strData.indexOf('#'));
    String cat  = strData.substring(strData.indexOf('#')+1, strData.indexOf('%'));
    atletas.put((String)numeros.elementAt(i), new Atleta(nome, Integer.parseInt(cat)));
    //System.out.println(nome + " " + cat); //Para fins de debug
  }
  return true;
}
```

Figura 20 - Métodos ‘addNumeros()’ e ‘addAtletas()’ da classe BaseDeDados.

4.4.2 Cronômetro

A opção “Cronômetro” é a parte principal do aplicativo e deve ser utilizada durante o andamento da competição. Através dela é possível acionar o cronômetro (tecla ‘START’) da aplicação e iniciar o “controle” da competição. O *display* do dispositivo passa a apresentar o cronômetro em funcionamento, um campo através do qual é possível “entrar” com o número do atleta e um segundo campo onde aparece o número, nome, tempo final, categoria, colocação geral e colocação por categoria do último atleta registrado no sistema ao cruzar a linha de chegada – o objetivo principal desse trabalho. A figura 21 apresenta o método

‘processInput()’, da classe Cronometro, responsável por registrar o tempo e as colocações geral e na categoria do atleta cujo número foi registrado no sistema.

```
private void processInput()
{
  Atleta atl = null;
  String key = tfEnter.getString();
  atl = base.removeAtleta(key);
  if(atl != null && atl.getColGeral() == -1)
  {
    atl.setTempo(tfClock.getString().substring(7));
    atl.setColGeral(cgCount);
    cgCount++;
    int cat = atl.getCat();
    atl.setColCategoria(ccCount[cat]);
    ccCount[cat]++;
    base.addAtleta(key, atl);
    ticker.setString("(" + key + ")" + atl.getNome() +
      "_" + atl.getNomeCat(cat) + "(" + atl.getColCategoria() + ")");
  }
}
```

Figura 21 - Método ‘processInput()’ da classe Cronômetro.

O campo contendo as informações do último atleta é visualizado na forma de um *ticker* (uma linha que se movimenta da esquerda para a direita até que toda a informação cruze a tela, repetindo esse processo até que um novo atleta cruze a linha de chegada, sendo o *ticker* atualizado com os dados deste novo atleta), pois o tamanho do *display* dos dispositivos móveis é geralmente restrito, não sendo possível imprimir todas as informações simultaneamente na tela. A figura 22 ilustra esta situação.

Figura 22 - Tela principal do MIDlet mostrando os dados de um atleta no *ticker*.

Ao término da competição, a tecla “STOP” é utilizada para parar o cronômetro. A tecla “MENU” retorna ao *menu* principal.

4.4.3 Exportar Dados

A opção “Exportar Dados” atualiza o banco de dados do servidor com o tempo total de prova, a classificação geral e a classificação por categoria de cada atleta. Esse processo é realizado de maneira similar à importação de dados: o aplicativo acessa, através do método ‘export()’ da classe BaseDeDados, a página ‘import.jsp’ utilizando como parâmetro o número do atleta, seu tempo final de prova, colocação geral e colocação na categoria. Esse processo é repetido para cada atleta participante. A figura 23 apresenta o método ‘export()’.

```
public void exportData()
{
  for(int i=0; i<numeros.size(); i++)
  {
    String StrTempNum = (String)numeros.elementAt(i);
    Atleta atl = (Atleta) atletas.get(StrTempNum);
    try
    {
      conexao.getsetData("import.jsp?num=" + StrTempNum + "&tempo=" +
        atl.getTempo() + "&cg=" + atl.getColGeral() + "&cc=" + atl.getColCategoria());
    }
    catch (Exception e)
    {
      Mensagem.showMessage("Mensagem...", "Erro na exportação de dados!");
    }
  }
  Mensagem.showMessage("Mensagem...", "Exportação de dados realizada com sucesso!");
}
```

Figura 23 - Método ‘export()’ da classe BaseDeDados.

4.4.4 Salvar Dados

Os dados da competição contidos no dispositivo de controle (PDA ou aparelho de telefonia celular) são mantido na memória volátil do aparelho. Isso significa que, se o mesmo for desligado, os dados serão perdidos. A opção ‘Salvar Dados’ pode ser utilizada para armazenar esses dados na memória não-volátil do aparelho através do *Record Management*

System (RMS), uma API de persistência de dados presente no MIDP. Assim, é criado um *Record Store* contendo os objetos 'atletas' e seus atributos. Dessa forma os dados da competição permanecem no dispositivo até que a opção 'Salvar Dados' seja utilizada novamente, sendo os dados antigos substituídos pelos novos dados salvos. O método 'saveBase()' da classe BaseDeDados, apresentado na figura 24, implementa esse processo.

```
public void saveBase()
{
  openRecStore();
  try
  {
    ByteArrayOutputStream strmBytes = new ByteArrayOutputStream();
    DataOutputStream strmDataType = new DataOutputStream(strmBytes);
    byte[] record;
    for(int i=0; i<numeros.size(); i++)
    {
      String StrTempNum = (String)numeros.elementAt(i);
      Atleta atl = (Atleta) atletas.get(StrTempNum);
      strmDataType.writeUTF(StrTempNum);
      strmDataType.writeUTF(atl.getNome());
      strmDataType.writeInt(atl.getCat());
      strmDataType.writeUTF(atl.getTempo());
      strmDataType.writeInt(atl.getColGeral());
      strmDataType.writeInt(atl.getColCategoria());
      strmDataType.flush();
      record = strmBytes.toByteArray();
      rs.addRecord(record, 0, record.length);
      strmBytes.reset();
    }
    strmBytes.close();
    strmDataType.close();
    numeros.removeAllElements();
    atletas.clear();
  }
  catch (Exception e)
  {
    e.printStackTrace(); //Para fins de debug
  }
  closeRecStore();
  Mensagem.showMessage("Mensagem...", "Salvamento de dados realizado com sucesso!");
}
```

Figura 24 - Método 'saveBase()' da classe BaseDeDados.

Essa opção é particularmente importante no caso de os dados da competição não poderem ser exportados para o banco de dados imediatamente, devido à falha temporária na conexão ou no servidor.

4.4.5 Recuperar Dados

A opção ‘Recuperar Dados’ é utilizada para recuperar os dados armazenados no dispositivo através da opção ‘Salvar Dados’, caso o dispositivo tenha sido desligado antes que o banco de dados do servidor pudesse ser atualizado. Os dados são passados da memória não-volátil para memória volátil, podendo, então, ser exportados para o servidor através da opção ‘Exportar Dados’. A figura 25 apresenta o método ‘restoreBase()’ da classe BaseDeDados, utilizado para implementar esse procedimento.

```
public void restoreBase()
{
  openRecStore();
  try
  {
    byte[] recData = new byte[100];
    ByteArrayInputStream strmBytes = new ByteArrayInputStream(recData);
    DataInputStream strmDataType = new DataInputStream(strmBytes);
    for(int i=1; i<=rs.getNumRecords(); i++)
    {
      rs.getRecord(i, recData, 0);
      String StrTempNum = strmDataType.readUTF();
      numeros.addElement(StrTempNum);
      atletas.put(StrTempNum, new Atleta(strmDataType.readUTF(),
      strmDataType.readInt(), strmDataType.readUTF(), strmDataType.readInt(),
        strmDataType.readInt()));
      strmBytes.reset();
    }
    strmBytes.close();
    strmDataType.close();
  }
  catch (Exception e)
  {
    e.printStackTrace(); //Para fins de debug
  }
  closeRecStore();
  Mensagem.showMessage("Mensagem...", "Recuperação de dados realizada com sucesso!");
}
```

Figura 25 - Método ‘restoreBase()’ da classe BaseDeDados.

4.4.6 Sobre

A opção “Sobre” traz algumas informações adicionais sobre a finalidade do presente aplicativo e alguns dados sobre os autores do mesmo.

5 CONCLUSÃO

Diante do quadro analisado antes da construção do sistema apresentado neste trabalho, haviam apenas dois tipos de sistemas utilizados para o gerenciamento e administração de competições de *triathlon*: o que faz uso de um *microchip* de controle e outro conhecido como "controle manual com pulseira".

O objetivo deste projeto foi desenvolver um sistema capaz de minimizar as grandes desvantagens do sistema de "controle manual com pulseira", buscando uma aproximação da eficiência do sistema de controle com *microchip*, ainda que mantendo o custo do novo sistema o mais baixo possível. Em síntese, o objetivo era desenvolver um sistema capaz de identificar cada atleta que cruza a linha de chegada, divulgando, "em tempo real", seu tempo total de prova, sua colocação geral e sua colocação na categoria, sendo, também, capaz de "gerar", poucos minutos após a chegada do último competidor, uma lista contendo nome, colocação, parciais de natação, ciclismo, corrida e tempo total de prova de todos os atletas participantes que completaram a competição (relatório de prova), e que tenha um custo de implementação baixo.

O sistema foi dividido em duas etapas: uma delas rodando no dispositivo, responsável pela captura das parciais durante a competição, e a outra rodando em um servidor, que contém um banco de dados onde é feito o cadastro dos atletas participantes da prova. As duas partes do sistema se comunicam através do protocolo HTTP, através do qual o dispositivo móvel acessa o banco de dados do servidor por intermédio de páginas *web* dinâmicas.

O objetivo principal proposto e alcançado nesse trabalho pode ser resumido na seguinte afirmação: através do novo sistema é possível a identificação do atleta assim que o mesmo cruza a linha de chegada (ou até mesmo antes disso), tornando possível, no mesmo

momento, a divulgação da sua colocação geral, colocação entre os atletas da sua categoria de idade e tempo final de prova.

O novo sistema mantém a característica de baixo custo por ser executável em uma variedade de dispositivos portáteis, tais como aparelhos de telefonia celular e PDAs, bastando para tanto que tais aparelhos possuam suporte à tecnologia J2ME.

O sistema desenvolvido registra apenas o tempo total de prova, a colocação geral e a colocação na categoria de cada atleta numa competição de *triathlon*. Uma sugestão para trabalhos futuros é a adaptação desse sistema de forma a serem registradas também as parciais de natação, transição 1, ciclismo, transição 2 e corrida. Para tanto é preciso alterar a estrutura do banco de dados, adicionando uma coluna para cada uma das novas parciais na tabela 'participantes'. Cada dispositivo de controle responsável pela captura de cada uma das parciais deve possuir instalado o MIDlet 'TriathlonApp'. Como todos os dispositivos devem ter seus cronômetros sincronizados, no momento da largada, a captura de cada parcial deve ser realizada sobre o tempo total de prova e a "filtragem" da parcial (o tempo real correspondente a cada parcial) deve ser feita no final da prova, subtraindo-se o tempo obtido no dispositivo pelo(s) tempo(s) registrado(s) pela(s) parcial(ais) anterior(es). Essa "filtragem" deve ser realizada no momento entre a exportação dos dados do dispositivo e a atualização do banco de dados no servidor, e pode ser feita na própria página JSP que gerencia esse processo.

6 REFERÊNCIAS

Almeida, L. B. **Introdução à J2ME e programação MIDP**, Mundo Java, Edição 5, Ano I, 2004, pp.20-27

Anselmo, F. **Tudo o Que Você Queria Saber Sobre o JSP Quando Utiliza o Servidor TomCat com o Banco MySQL**. Florianópolis: Visual Books, 2002.

Anselmo, F. **Tudo o que você queria saber sobre a JDBC... mas ninguém quis (ou não sabia) responder**. Florianópolis: Visual Books, 2001.

Championchip, disponível em www.championchip.com.br, acessado em Junho/2004

Chiptiming, disponível em www.chiptiming.com.br, acessado em Junho/2004

Chronomix Corp, disponível em www.chronomix.com, acessado em Junho/2004

Deitel, H. M. & Deitel, P. J. **Java: Como Programar**, Porto Alegre: Bookman, 4ª edição, 2002.

Hyalen, C.M. **Java 2 Micro Edition: do *pager* ao PDA, com J2ME**, Java Magazine, Edição 1, Ano 1, pp. 32-35.

Martins, F. Record Store: o pacote javax.microedition.rms, disponível em www.cpda.com.br, acessado em Junho/2004.

Miranda, C. **J2ME no JavaOne 2002: multimídia e jogos no Java Micro Edition**, Java Magazine, Edição 1, Ano 1, pp. 16-19.

Miranda, C. **Conectividade com MIDP: interoperabilidade com serviços externos**, Java Magazine, Edição 6, Ano 1, pp. 22-26.

Miranda, C. **Multimídia no celular: Mobile Media API (MMAPI)**, Java Magazine, Edição 2, Ano 1, pp. 24-26

Miranda, C. **Dados em J2ME: persistência em dispositivos com RMS e JDBC**, Java Magazine, Edição 3, Ano 1, pp. 20-23.

Miranda, C. **A outra face do J2ME: serviços e novas APIs de intra-estrutura wireless**, Java Magazine, Edição 4, Ano 1, pp. 26-28.

Montenegro, C. **Uso do J2ME em uma aplicação de CRM**, Mundo JAVA, número 2, ano 1, pp. 25-28.

Muchow, J. W. ***Core J2ME: Technology & MIDP***. New Jersey: Prentice Hall PTR, 2001.

Reese, G. *et all*. ***Managing & Using MySQL***. Sebastopol: O'Reilly & Associates, Second Edition, 2002.

Sabino, V. **Game API: Simplicidade e poder em jogos para celulares**, in Java Magazine, Edição 10, Ano 1, pp. 42-43.

Souza, B. **Fazendo wireless acontecer: J2ME e o mercado brasileiro**, in Java Magazine, Edição 3, Ano 1, pp. 12-15.

Topley, K. ***J2ME in a Nutshell***. Sebastopol: O'Reilly & Associates, First Edition, 2002.

7 ANEXOS

Código no servidor

acesso.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     acesso.jsp
     ------------
        Página JSP utilizada por todas as demais páginas JSP que acessam o
        banco de dados. Ela é responsável por indicar o driver através do
        qual a linguagem Java se comunica com o banco de dados MySQL bem como
        por fornecer o local onde se encontra e o nome da base de dados, além
        da porta utilizada para comunicação, username e senha de identificação
        do usuário.
-%>

<%@page import="java.sql.*"%>
<%
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/triathlon","root","");
Statement stm = con.createStatement();
%>
```

listagem.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     listagem.jsp
     ------------
        Página JSP principal da interface com o usuário através da qual é possível
        cadastrar os atletas no banco de dados do sistema (MySQL), visualizar e fazer
        modificações no mesmo,
-%>


<%@page contentType="text/html; charset=ISO-8859-1" language="java"
import="java.sql.*,java.io.*"%>
<html>
<head>
 <title>Cadrastro de Atletas</title>
 <script language="JavaScript">

   function exclui(idt) {
     if (window.confirm("Deseja realmente a exclusão deste atleta?"))
       document.location.href="regexc.jsp?num=" + idt;
   }

   function altera(idt) {
     document.location.href="altera.jsp?num=" + idt;
   }



    //gera lista de atletas inscritos
   function geraListaInscritos() {
     document.location.href="inscritos.jsp";
```

```
   }

    //gera lista com os resultados
   function geraResultados() {
     document.location.href="resultados.jsp";

   }



 </script>
</head>

<body>
<%
response.setDateHeader("Expires", 0);
response.setHeader("Pragma", "no-cache");
if (request.getProtocol().equals("HTTP/1.1"))
  response.setHeader("Cache-Control", "no-cache");
%>

<%@include file="conecta/acesso.jsp"%>

<%
ResultSet res = stm.executeQuery("select max(numero) from participantes");
int maior;
if (res.next())
  maior = res.getInt(1)+1;
else
  maior = 1;
res.close();
%>

<FORM action='reginc.jsp' method=post name=form1>
<h3>Cadastro de Atletas</h3>
<TABLE>
  <TR>
    <TD>Número:</TD>
    <TD><input type=text name=numero value='<%=maior%>' maxlength='5' size='5' readonly></TD>
  </TR>
  <TR>
    <TD>Nome:</TD>
    <TD><input type=text value='' name=nome maxlength='50' size='50'></TD>
  </TR>
  <TR>
    <TD>Sexo:</TD>
    <TD>
      <input type=radio name=sexo value=M checked />Masculino  
      <input type=radio name=sexo value=F />Feminino  
    </TD>
  </TR>



  <TR>
    <TD>Categoria:</TD>
      <TD>
        <select name=categoria>
          <option value=Elite>Elite</option>
          <option value=12-14>12 à 14 anos</option>
          <option value=15-16>15 à 16 anos</option>
          <option value=17-19>17 à 19 anos</option>
          <option value=20-24>20 à 24 anos</option>
          <option value=25-29>25 à 29 anos</option>
          <option value=30-34>30 à 34 anos</option>
          <option value=35-39>35 à 39 anos</option>
          <option value=40-44>40 à 44 anos</option>
          <option value=45-49>45 à 49 anos</option>
          <option value=50-54>50 à 54 anos</option>
          <option value=55-59>55 à 59 anos</option>
          <option value=60-64>60 à 64 anos</option>
          <option value=65-69>65 à 69 anos</option>
          <option value=70-74>70 à 74 anos</option>
          <option value=75-79>75 à 79 anos</option>
          <option value=80+++>acima de 80 anos</option>
        </select>
```

```
      </TD>
      <TD>
        <p align=center><input type=submit value=' Incluir '></p>
      </TD>

  </TR>
  <TR>

</TABLE>

</FORM>

<hr>



<h3>Atletas já cadastrados:</h3>
<TABLE width="60%">
  <tr bgcolor='#009090'>
    <td align=center width=30><b>Nº</b></td>
    <td width=500><b>Nome</b></td>
    <td width=100><b>Categoria</b></td>
    <td align=center><b>Manutenção</b></td>
  </tr>

 <%
 res = stm.executeQuery("select numero, nome, categoria " + " from participantes order by
nome");
 String num;
 boolean liga = true;
 while (res.next()) {
    out.println("<tr bgColor=" + ((liga = !liga)?"#00B0B0":"") + ">");
    num = res.getString("numero");
    out.println("<td align=center>" + num + "</td>");
    out.println("<td>" + res.getString("nome") + "</td>");
    out.println("<td>" + res.getString("categoria") + "</td>");
    out.println("<td align=center>" +
      "<input type=button onClick='javaScript:altera(" + num + ");'" +
      "style='cursor:hand;' border=0 value='Alterar'>" +
      "<input type=button onClick='javaScript:exclui(" + num + ");'" +
      "style='cursor:hand;' border=0 value='Eliminar'>" +
      "</td>");
    out.println("</tr>");
 }
 %>

 </TABLE>
  <%
     res = stm.executeQuery("select count(*) from participantes");
       int total=0;
       if (res.next()) {
       total=res.getInt("count(*)");
     }
     out.println("<TOTAL>Total de atletas cadastrados: " + total +"</TOTAL>");
  %>
<FORM>
  <p align=left><input type=button onClick='javaScript:geraListaInscritos();'
style='cursor:hand' value=' Gera Lista Inscritos '></p>
  <p align=left><input type=button onClick='javaScript:geraResultados();' style='cursor:hand'
value=' Gera Lista Resultados '></p>
</FORM>
 </br><hr>
 </body>
 </html>
```

altera.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     altera.jsp
     ------------
       Página JSP utilizada para alterar os dados de um atleta previamente
       registrado.
-%>
```

```
<%@page contentType="text/html; charset=ISO-8859-1" language="java" import="java.sql.*"%>
<html>
<head>
 <title>Cadrastro de Títulos</title>
</head>

<body>
<%@include file="conecta/acesso.jsp"%>

<h3>Alteração de Titulos</h3>
<FORM action='regnmod.jsp' method=post name=form1>

<%
ResultSet res = stm.executeQuery("select nome, sexo, categoria from participantes where
numero=" +
  request.getParameter("num"));
if (res.next()) {
%>

<TABLE>
  <TR>
    <TD>Número:</TD>
    <TD><input type=text name=numero value='<%=request.getParameter("num")%>' maxlength='5'
size='5' readonly></TD>
  </TR>
  <TR>
    <TD>Nome:</TD>
    <TD><input type=text value='<%=res.getString("nome")%>' name=nome maxlenght='50'
size='50'></TD>
  </TR>
  <TR>
      <TD>Sexo:</TD>
       <TD>
       <input type=radio name=sexo value=M checked />Masculino  
       <input type=radio name=sexo value=F />Feminino  
       </TD>
  </TR>
  <TR>
      <TD>Categoria:</TD>
       <TD>
               <select name=categoria>
                 <option value=Elite>Elite</option>
                 <option value=12-14>12 à 14 anos</option>
                 <option value=15-16>15 à 16 anos</option>
                 <option value=17-19>17 à 19 anos</option>
                 <option value=20-24>20 à 24 anos</option>
                 <option value=25-29>25 à 29 anos</option>
                 <option value=30-34>30 à 34 anos</option>
                 <option value=35-39>35 à 39 anos</option>
                 <option value=40-44>40 à 44 anos</option>
                 <option value=45-49>45 à 49 anos</option>
                 <option value=50-54>50 à 54 anos</option>
                 <option value=55-59>55 à 59 anos</option>
                 <option value=60-64>60 à 64 anos</option>
                 <option value=65-69>65 à 69 anos</option>
                 <option value=70-74>70 à 74 anos</option>
                 <option value=75-79>75 à 79 anos</option>
                 <option value=80+++>acima de 80 anos</option>
               </select>
      </TD>
  </TR>
</TABLE>
<p align=center><input type=submit value=' Alterar '></p>
</FORM>

<%
}
res.close();
%>

</body>
</html>
```

numeros.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     numeros.jsp
     ------------
       Página JSP acessada pelo aplicativo MIDlet e que retorna o número de
       cada um dos atletas registrados no banco de dados.
-%>

<%@page contentType="text/html; charset=ISO-8859-1" language="java" import="java.sql.*"%>
<%@include file="conecta/acesso.jsp"%>
<%ResultSet res = stm.executeQuery("select numero from participantes");
  int a=0; while (res.next()){a=res.getInt("numero");%>
 <%=a%><%}%>
 <%res.close();%>
```

quote.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     quote.jsp
     ------------
       Página JSP que utiliza um número como parâmetro e através dele realiza
       uma pesquisa no banco de dados, retornando o nome e a categoria do
       atleta de posse desse número.
-%>


<%@page contentType="text/html; charset=ISO-8859-1" language="java" import="java.sql.*"%>
<%@include file="conecta/acesso.jsp"%>
<%
ResultSet res = stm.executeQuery("select nome, catID from participantes where numero=" +
request.getParameter("num"));
if (res.next())
       out.println('*'+res.getString("nome")+'#'+res.getString("catID")+'%');
res.close();
%>
```

regexc.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     lregexc.jsp
     ------------
       Página JSP que apaga do banco de dados o atleta cujo número equivale
       ao número entrado como parâmetro.
-%>

<%@page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"%>

<%@include file="conecta/acesso.jsp"%>

<%
int res = stm.executeUpdate("delete from participantes where numero = " +
  request.getParameter("num"));

//Retorno
out.println("<SCRIPT language='JavaScript'>;");
if (res>0) {
  out.println("alert('Registro Excluído');");
}
else {
  out.println("alert('Registro nao Excluído');");
}
out.println("history.go(-1);");
out.println("</SCRIPT>");
%>
```

reginc.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     reginc.jsp
     ------------
       Página JSP que é chamada pela página listagem.jsp e que insere o novo
       no banco atleta no banco de dados.
-%>


<%@page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"%>
<%@include file="conecta/acesso.jsp"%>

<%
int catnum=-1;
String catID = request.getParameter("sexo")+request.getParameter("categoria");

if (catID.equals("MElite")) catnum=0;
else if (catID.equals("M12-14")) catnum=1;
else if (catID.equals("M15-16")) catnum=2;
else if (catID.equals("M17-19")) catnum=3;
else if (catID.equals("M20-24")) catnum=4;
else if (catID.equals("M25-29")) catnum=5;
else if (catID.equals("M30-34")) catnum=6;
else if (catID.equals("M35-39")) catnum=7;
else if (catID.equals("M40-44")) catnum=8;
else if (catID.equals("M45-49")) catnum=9;
else if (catID.equals("M50-54")) catnum=10;
else if (catID.equals("M55-59")) catnum=11;
else if (catID.equals("M60-64")) catnum=12;
else if (catID.equals("M65-69")) catnum=13;
else if (catID.equals("M70-74")) catnum=14;
else if (catID.equals("M75-79")) catnum=15;
else if (catID.equals("M80+++")) catnum=16;
else if (catID.equals("FElite")) catnum=17;
else if (catID.equals("F12-14")) catnum=18;
else if (catID.equals("F15-16")) catnum=19;
else if (catID.equals("F17-19")) catnum=20;
else if (catID.equals("F20-24")) catnum=22;
else if (catID.equals("F25-29")) catnum=22;
else if (catID.equals("F30-34")) catnum=23;
else if (catID.equals("F35-39")) catnum=24;
else if (catID.equals("F40-44")) catnum=25;
else if (catID.equals("F45-49")) catnum=26;
else if (catID.equals("F50-54")) catnum=27;
else if (catID.equals("F55-59")) catnum=28;
else if (catID.equals("F60-64")) catnum=26;
else if (catID.equals("F65-69")) catnum=30;
else if (catID.equals("F70-74")) catnum=31;
else if (catID.equals("F75-79")) catnum=32;
else if (catID.equals("F80+++")) catnum=33;

//out.println("categoria: " + catID + ", catnum: " + catnum);



int res = stm.executeUpdate("insert into participantes(numero, nome, sexo, categoria, catID)"
+
  "values(" +request.getParameter("numero")+ ", '" +request.getParameter("nome") + "', '"
  +request.getParameter("sexo")+ "', '"
+request.getParameter("sexo")+request.getParameter("categoria")+ "', " +catnum+ ")" );


//Retorno
out.println("<SCRIPT language='JavaScript'>");
if (res>0) {
  out.println("alert('Registro incluído');");
}
else {
  out.println("alert('Registro não incluído');");
}
```

```
out.println("history.go(-1);");
out.println("</SCRIPT>");

%>
```

regnmod.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     recmod.jsp
     ------------
       Página JSP que é chamada pela página altera.jsp e efetua um "update"
       no banco de dados à partir das modificações nos dados do atleta
       realizada na página altera.jsp.
-%>


<%@page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"%>

<%@include file="conecta/acesso.jsp"%>

<%
int catnum=-1;
String catID = request.getParameter("sexo")+request.getParameter("categoria");

if (catID.equals("MElite")) catnum=0;
else if (catID.equals("M12-14")) catnum=1;
else if (catID.equals("M15-16")) catnum=2;
else if (catID.equals("M17-19")) catnum=3;
else if (catID.equals("M20-24")) catnum=4;
else if (catID.equals("M25-29")) catnum=5;
else if (catID.equals("M30-34")) catnum=6;
else if (catID.equals("M35-39")) catnum=7;
else if (catID.equals("M40-44")) catnum=8;
else if (catID.equals("M45-49")) catnum=9;
else if (catID.equals("M50-54")) catnum=10;
else if (catID.equals("M55-59")) catnum=11;
else if (catID.equals("M60-64")) catnum=12;
else if (catID.equals("M65-69")) catnum=13;
else if (catID.equals("M70-74")) catnum=14;
else if (catID.equals("M75-79")) catnum=15;
else if (catID.equals("M80+++")) catnum=16;
else if (catID.equals("FElite")) catnum=17;
else if (catID.equals("F12-14")) catnum=18;
else if (catID.equals("F15-16")) catnum=19;
else if (catID.equals("F17-19")) catnum=20;
else if (catID.equals("F20-24")) catnum=22;
else if (catID.equals("F25-29")) catnum=22;
else if (catID.equals("F30-34")) catnum=23;
else if (catID.equals("F35-39")) catnum=24;
else if (catID.equals("F40-44")) catnum=25;
else if (catID.equals("F45-49")) catnum=26;
else if (catID.equals("F50-54")) catnum=27;
else if (catID.equals("F55-59")) catnum=28;
else if (catID.equals("F60-64")) catnum=26;
else if (catID.equals("F65-69")) catnum=30;
else if (catID.equals("F70-74")) catnum=31;
else if (catID.equals("F75-79")) catnum=32;
else if (catID.equals("F80+++")) catnum=33;

//out.println("categoria: " + catID + ", catnum: " + catnum);

int res = stm.executeUpdate("update participantes set nome='" +request.getParameter("nome")+
    "', catID="+catnum+", sexo='" + request.getParameter("sexo")+ "', categoria='"
+request.getParameter("sexo")+request.getParameter("categoria")+
    "' where numero=" +request.getParameter("numero"));


//Retorno
out.println("<SCRIPT language='JavaScript'>;");
if (res>0) {
  out.println("alert('Registro Modificado');");
  out.println("history.go(-2);");
}
else {
```

```
  out.println("alert('Registro nao Modificado');");
  out.println("history.go(-1);");
}
out.println("</SCRIPT>");
%>
```

total.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     total.jsp
     ------------
       Página JSP que acessa o banco de dados e retorna o número total de
       atletas.
-%>

<%@page contentType="text/html; charset=ISO-8859-1" language="java" import="java.sql.*"%>

<%@include file="conecta/acesso.jsp"%>


<%
ResultSet res = stm.executeQuery("select count(*) from participantes");
int a=0;
if (res.next()) {
       a=res.getInt("count(*)");
}
%>

<%=a%>

<%
res.close();
%>
```

inscritos.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     inscritos.jsp
     ------------
       Página JSP que acessa o banco de dados e retorna uma lista com os
       atletas inscritos.
-%>

<%@page contentType="text/html; charset=ISO-8859-1" language="java"
import="java.sql.*,java.io.*"%>
<html>
<head>
 <title>Lista dos participantes</title>
</head>

<body>
<%
response.setDateHeader("Expires", 0);
response.setHeader("Pragma", "no-cache");
if (request.getProtocol().equals("HTTP/1.1"))
  response.setHeader("Cache-Control", "no-cache");
%>

<%@include file="conecta/acesso.jsp"%>


<h3>Relação de Atletas Inscritos:</h3>
<TABLE width="40%">
  <tr>
    <td width=50><b>Nº:</b></td>
    <td><b>Nome</b></td>
    <td width=100><b>Categoria</b></td>
  </tr>
```

```
 <%
 ResultSet res = stm.executeQuery("select numero, nome, categoria" + " from participantes
order by nome");
 while (res.next()) {
    out.println("<td>" + res.getString("numero") + "</td>");
    out.println("<td>" + res.getString("nome") + "</td>");
    out.println("<td>" + res.getString("categoria") + "</td>");
    out.println("</tr>");
 }
 %>

 </TABLE>
   <%
      res = stm.executeQuery("select count(*) from participantes");
        int total=0;
        if (res.next()) {
               total=res.getInt("count(*)");
      }
      out.println("<TOTAL>Total de atletas inscritos: " + total +"</TOTAL>");
  %>
 </br><hr>
 </body>
 </html>
```

---

resultados.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     resultados.jsp
     ------------
       Página JSP que acessa o banco de dados e retorna uma lista com os
       resultados da competição.
-%>

<%@page contentType="text/html; charset=ISO-8859-1" language="java"
import="java.sql.*,java.io.*"%>
<html>
<head>
 <title>Resultados</title>
</head>

<body>
<%
response.setDateHeader("Expires", 0);
response.setHeader("Pragma", "no-cache");
if (request.getProtocol().equals("HTTP/1.1"))
  response.setHeader("Cache-Control", "no-cache");
%>

<%@include file="conecta/acesso.jsp"%>


<h3>Resultados:</h3>
<TABLE width="60%">
  <tr>
    <td width=50><b>CG</b></td>
    <td width=50><b>CC</b></td>
    <td width=50><b>Nº</b></td>
    <td><b>Nome</b></td>
    <td width=100><b>Categoria</b></td>
    <td width=70><b>Tempo</b></td>
  </tr>

 <%
 ResultSet res = stm.executeQuery("select numero, nome, categoria, CG, CC, tempo" + " from
participantes order by CG");
 while (res.next()) {
    out.println("<td>" + res.getString("CG") + "</td>");
    out.println("<td>" + res.getString("CC") + "</td>");
    out.println("<td>(" + res.getString("numero") + ")</td>");
    out.println("<td>" + res.getString("nome") + "</td>");
```

```
    out.println("<td>" + res.getString("categoria") + "</td>");
    out.println("<td>" + res.getString("tempo") + "</td>");
    out.println("</tr>");
 }
 %>

 </TABLE>
 </br><hr>
 </body>
 </html>
```

atualiza.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     atualiza.jsp
     ------------
       Página JSP utilizada pelo MIDlet para atualizar o banco de dados com os
      novos dados provenientes da competição.
-%>

<%@page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"%>

<%@include file="conecta/acesso.jsp"%>

<%
int res = stm.executeUpdate("update participantes set CG='" +request.getParameter("cg")+
    "', CC='" + request.getParameter("cc")+ "'
    where numero=" +request.getParameter("numero"));

//Retorno
out.println("<SCRIPT language='JavaScript'>;");
if (res>0) {
  out.println("alert('Registro Alterado');");
  out.println("history.go(-2);");
}
else {
  out.println("alert('Registro nao Alterado');");
  out.println("history.go(-1);");
}
out.println("</SCRIPT>");
%>
```

import.jsp

```
<%- "Sistema para gerenciamento e administração de competições de triathlon"

     Autores: Denis Hauffe e Fernando Laudares

     import.jsp
     ------------
       Página JSP utilizada pelo MIDlet para importar a lista com a relação
     dos atletas inscritos do Banco de Dados.
-%>

<%@page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"%>

<%@include file="conecta/acesso.jsp"%>

<%


int res = stm.executeUpdate("update participantes set CG=" +request.getParameter("cg")+
          ", CC=" +request.getParameter("cc")+
          ", tempo='" +request.getParameter("tempo")+
          "' where numero=" +request.getParameter("num"));


%>
```

Código no dispositivo móvel

TriathlonAppMain.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       TriathlonAppMain.java
*       ---------------------
*                       Classe principal do pacote J2ME que estende a classe genérica
*               MIDlet (esse é o aplicativo MIDlet do pacote 'TriathlonApp'.
*               Sendo um MIDlet, ele é obrigado a implementar o construtor
*               genérico da classe e os métodos startApp(), pauseApp(),
*               destroyApp() e quitApp(), ainda que os mesmos não façam nada.
*                       O que essa classe faz é instanciar um objeto da classe
*               'MenuPrincipal' e ceder o display (tela do dispositivo) para ele.
*/

package TriathlonApp;


import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;



public class TriathlonAppMain extends MIDlet
{
  static TriathlonAppMain instance;
  MenuPrincipal menu;

  public TriathlonAppMain()
  {
    instance = this;
    menu = new MenuPrincipal();
  }

  protected void startApp()
  {
    Display.getDisplay(this).setCurrent(menu);
  }

  protected void pauseApp()
  {
  }

  protected void destroyApp(boolean u)
  {
  }

  public static void quitApp()
  {
    instance.destroyApp(true);
    instance.notifyDestroyed();
    instance = null;
  }
}
```

MenuPrincipal.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       MenuPrincipal.java
*       ---------------------
*                       Classe que implementa o "CommandListener" do J2ME e que define
*               o Menu principal do aplicativo, com as opções.
*                       Ela também instacia um objeto da classe BaseDeDados e outro da classe
*               Cronometro.                                                              */
```

```
package TriathlonApp;


import javax.microedition.lcdui.*;


public class MenuPrincipal extends List implements CommandListener
{
  // Opções do nosso menu principal
  private static final String[] MENU_CHOICES = new String[6];
  static
  {
       MENU_CHOICES[0] = "Importar Dados";
    MENU_CHOICES[1] = "Cronômetro";
    MENU_CHOICES[2] = "Exportar Dados";
    MENU_CHOICES[3] = "Salvar Dados";
    MENU_CHOICES[4] = "Recuperar Dados";
    MENU_CHOICES[5] = "Sobre";
  };
  private BaseDeDados base = new BaseDeDados();
  private Cronometro cronometro = new Cronometro(this, base);



  public MenuPrincipal()
  {
    super("TriathlonApp", List.IMPLICIT, MENU_CHOICES, null);
    try
    {
      init();
    }
    catch(Exception e)
    {
      e.printStackTrace(); // Para fins de debug
    }
  }

  private void init() throws Exception
  {
    // Set up do Displayable para ouvir eventos de comandos
    setCommandListener(this);
    // Adiciona o comando Sair
    addCommand(new Command("Sair", Command.EXIT, 1));
    // Define o endereço (URL) da parte do servidor da aplicação
    base.setURL("http://localhost:8080/testeJSP/gerencia/");
  }


  public void commandAction(Command command, Displayable displayable)
  {
    // Recebe o comando requisitado pelo usuário
    int commandType = command.getCommandType();

    if (commandType == Command.EXIT)
    {
      ConfirmacaoSair cfm = new ConfirmacaoSair(this);
    }
    else
    {
      String selectedItem = getString(getSelectedIndex());
      if (selectedItem.equals(MENU_CHOICES[0]))
      {
       // Apaga algun registro de persistência se houver
       base.deleteRecStore();
       // Importa a base de dados
       base.importData();
       base.showDataBase(); // Para fins de debug
      }
      else if (selectedItem.equals(MENU_CHOICES[1]))
      {
       // Vai para a tela do cronômetro
        Display.getDisplay(TriathlonAppMain.instance).setCurrent(cronometro);
      }
      else if (selectedItem.equals(MENU_CHOICES[2]))
      {
       // Exporta a base de dados
        base.exportData();
```

```
      }
      else if (selectedItem.equals(MENU_CHOICES[3]))
      {
       // Salva a base de dados
       base.saveBase();
      }
      else if (selectedItem.equals(MENU_CHOICES[4]))
      {
       // Recupera a base de dados
        base.restoreBase();
        base.showDataBase(); // Para fins de debug
      }
      else if (selectedItem.equals(MENU_CHOICES[5]))
      {
       // Mostra o a tela "Sobre"
           Sobre.showAbout();
      }
    }
  }


}
```

Cronometro.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       Cronometro.java
*       ---------------------
*                     Classe que também implementa o CommandListener e que é derivada
*              da classe form.
*/

package TriathlonApp;


import java.util.Timer;
import java.util.TimerTask;
import javax.microedition.lcdui.*;



public class Cronometro extends Form implements CommandListener
{
  private Timer timer; // Timer utilizado para rodar o cronometro
  private RodaCronometro rodaCronometro; // Instância de RodaCronometro que
                                         // extende a classe TimerTask
  private long startTime = 0;
  private int cgCount;
  private int ccCount[] = new int[34];
  private TextField tfClock = new TextField(null,"       0:00:00.0", 20, TextField.ANY);
  private TextField tfEnter = new TextField("    Número:", null, 3, TextField.NUMERIC);
  private Command enterCommand = new Command("Entra", Command.OK, 1);
  private Command startCommand = new Command("Start", Command.SCREEN, 1);
  private Command stopCommand  = new Command("Stop", Command.SCREEN, 1);
  private Command menuCommand  = new Command("Menu", Command.SCREEN, 2);
  private Ticker ticker = new Ticker (".....");
  private MenuPrincipal menu;
  private BaseDeDados base;


  public Cronometro(MenuPrincipal menu, BaseDeDados base)
  {
       super(null);
    try
    {
      this.menu = menu;
      this.base = base;
      init();
    }
    catch(Exception e)
```

```
  {
    e.printStackTrace(); // Para fins de debug
  }
}


private void init() throws Exception
{
     append(tfEnter);
     append(tfClock);
     addCommand(menuCommand);
  addCommand(startCommand);
  setTicker(ticker);
  setCommandListener(this);
}


public void commandAction(Command command, Displayable displayable)
{
  if (command == startCommand)
  {
    // Captura o tempo atual
    startTime = System.currentTimeMillis();
    // Cria um novo objeto Timer para rodar o cronometro
    timer = new Timer();
    // Cria um novo objeto RodaCronometro para ser usado pelo Timer
    rodaCronometro = new RodaCronometro();
    // Requisita ao timer que rode o cronometro
    timer.scheduleAtFixedRate(rodaCronometro,
                0,   // Quantos milisegundos de delay antes de rodar
                10); // Tempo em milisegundos entre sucessivas execuções
                     // (i.e. roda a cada 10 milisegundos).
    cgCount = 1;
    for(int i=0; i<34; i++)
      ccCount[i] = 1;
    removeCommand(startCommand);
    addCommand(enterCommand);
    removeCommand(menuCommand);
    addCommand(stopCommand);
  }
  else if (command == stopCommand)
  {
    ConfirmacaoParar cfm = new ConfirmacaoParar(this);
  }
  else if (command == menuCommand)
  {
    // Volta para tela do menu principal
    Display.getDisplay(TriathlonAppMain.instance).setCurrent(menu);
  }
  else if (command == enterCommand)
  {
    if(timer != null)
    {
      processInput();
      tfEnter.setString("");
    }
  }
}


private void processInput()
{
     Atleta atl = null;
  String key = tfEnter.getString();
  atl = base.removeAtleta(key);
  if(atl != null && atl.getColGeral() == -1)
  {
    atl.setTempo(tfClock.getString().substring(7));
       atl.setColGeral(cgCount);
    cgCount++;
    int cat = atl.getCat();
    atl.setColCategoria(ccCount[cat]);
    ccCount[cat]++;
    base.addAtleta(key, atl);
    ticker.setString("(" + key + ")" + atl.getNome() + "_" + atl.getNomeCat(cat)
                                          + "(" + atl.getColCategoria() + ")");
  }
```

```
  }


  public void processStop()
  {
       timer.cancel(); // Pára o timer
    timer = null;
    removeCommand(enterCommand);
    addCommand(menuCommand);
    removeCommand(stopCommand);
  }


  // Formata o número dado em milisegundos para o formato "MM:SS:T"
  private String formatTime(long milliseconds) {
    long horas = 0; // Começa com zero horas
    long minutes = 0;
    long seconds = milliseconds / 1000;
    long tenths  = (milliseconds / 100) % 10;
    // Verifica a quantidade de segundos
    if (seconds >= 60)
    {
      // Se temos mais de 60 segundos, passamos para minutos
      minutes = seconds / 60;
      seconds = seconds % 60;
    }

    if (minutes >= 60)
    {
      horas = minutes / 60;
      minutes = minutes % 60;
    }

    String horaString = String.valueOf(horas);

    // Cria nossa string de minutos
    String minuteString = String.valueOf(minutes);
    if (minutes < 10)
    {
      minuteString = "0" + minuteString;
    }

    // Cria nossa string de segundos
    String secondsString = String.valueOf(seconds);
    if (seconds < 10)
    {
      secondsString = "0" + secondsString;
    }

    // Retorna tudo junto numa string só
    return "       " + horaString + ":" + minuteString + ":" + secondsString
                                              + "." + String.valueOf(tenths);
  }


  // É executada a cada task do nosso timer ou seja a cada 10 milisegundos
  class RodaCronometro extends TimerTask
  {
    public void run()
    {
      long elapsed = (startTime == 0)
           ? 0
           : System.currentTimeMillis() - startTime;
      tfClock.setString(formatTime(elapsed));
    }
  }

}
```

Atleta.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       Atleta.java
*       ----------------------
*                       Classe que define o objeto atleta.
*/

package TriathlonApp;


public class Atleta
{
  private String nome;
  private int categoria;
  private String tempo;
  private int colGeral;
  private int colCategoria;


  public Atleta(String nome, int categoria)
  {
        this.nome = nome;
        this.categoria = categoria;
        this.tempo = "";
        this.colGeral = -1;
        this.colCategoria = -1;
  }


  public Atleta(String nome, int categoria, String tempo, int colGeral, int colCategoria)
  {
        this.nome = nome;
        this.categoria = categoria;
        this.tempo = tempo;
        this.colGeral = colGeral;
        this.colCategoria = colCategoria;
  }


  public void setTempo(String tempo)
  {
        this.tempo = tempo;
  }


  public void setColGeral(int colGeral)
  {
        this.colGeral = colGeral;
  }


  public void setColCategoria(int colCategoria)
  {
        this.colCategoria = colCategoria;
  }


  public String getNome()
  {
    return nome;
  }


  public int getCat()
  {
        return categoria;
  }


  public String getNomeCat(int cat)
  {
        switch(cat)
        {
```

```
            case 0 : return "MElite";
        case 1 : return "M12-14";
        case 2 : return "M15-16";
        case 3 : return "M17-19";
        case 4 : return "M20-24";
        case 5 : return "M25-29";
        case 6 : return "M30-34";
        case 7 : return "M35-39";
        case 8 : return "M40-44";
        case 9 : return "M45-49";
        case 10: return "M50-54";
        case 11: return "M55-59";
        case 12: return "M60-64";
        case 13: return "M65-69";
        case 14: return "M70-74";
        case 15: return "M75-79";
        case 16: return "M80+";
        case 17: return "FElite";
        case 18: return "F12-14";
        case 19: return "F15-16";
        case 20: return "F17-19";
        case 21: return "F20-24";
        case 22: return "F25-29";
        case 23: return "F30-34";
        case 24: return "F35-39";
        case 25: return "F40-44";
        case 26: return "F45-49";
        case 27: return "F50-54";
        case 28: return "F55-59";
        case 29: return "F60-64";
        case 30: return "F65-69";
        case 31: return "F70-74";
        case 32: return "F75-79";
        case 33: return "F80+";
      }
      return null;
    }


    public String getTempo()
    {
         return tempo;
    }


    public int getColGeral()
    {
         return colGeral;
    }


    public int getColCategoria()
    {
         return colCategoria;
    }


}
```

BaseDeDados.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       BaseDeDados.java
*       ----------------------
*                       Classe que define a base de dados, cujos dados são importados do
*               banco de dados MySQL no servidor.
*/


package TriathlonApp;
```

```
import java.io.*;
import java.util.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;


public class BaseDeDados
{
  private Vector numeros;
  private Hashtable atletas;
  private RecordStore rs = null; // Record store
  static final String REC_STORE = "db_1"; // Nome do record store
  private Conexao conexao;


  public BaseDeDados()
  {
       numeros = new Vector();
       atletas = new Hashtable();
       conexao = new Conexao();
  }


  public void setURL(String url)
  {
       conexao.setURL(url);
  }


  public void importData()
  {
       if (addNumeros() && addAtletas())
         Mensagem.showMessage("Mensagem...", "Importação de dados realizada com sucesso!");
       else
         Mensagem.showMessage("Mensagem...", "Erro na importação de dados!");
  }


  public void exportData()
  {
       for(int i=0; i<numeros.size(); i++)
       {
         String StrTempNum = (String)numeros.elementAt(i);
      Atleta atl = (Atleta) atletas.get(StrTempNum);
      try
      {
        conexao.getsetData("import.jsp?num=" + StrTempNum + "&tempo="
        + atl.getTempo() + "&cg=" + atl.getColGeral() + "&cc=" + atl.getColCategoria());
      }
      catch (Exception e)
      {
       Mensagem.showMessage("Mensagem...", "Erro na exportação de dados!");
      }
       }
       Mensagem.showMessage("Mensagem...", "Exportação de dados realizada com sucesso!");
  }


  private boolean addNumeros()
  {
       int i;
       String strNum = null;
       String strInsc = null;
       try
       {
         strInsc = conexao.getsetData("numeros.jsp");
    }
    catch(Exception e)
    {
      e.printStackTrace(); // Para fins de debug
    }
    i = strInsc.indexOf('*');
    strInsc = strInsc.substring(i+1);
    while(strInsc.length() > 1)
    {
      i = strInsc.indexOf(' ');
```

```
    strNum = strInsc.substring(0, i);
    numeros.addElement(strNum);
    strInsc = strInsc.substring(i+1);
    //System.out.println(strNum); // Para fins de debug
  }
  return true;
}


private boolean addAtletas()
{
     String strData = null;
  for(int i=0; i<numeros.size(); i++)
     {
       try
       {
         strData = conexao.getsetData("quote.jsp?num=" + (String)numeros.elementAt(i));
    }
    catch(Exception e)
    {
     e.printStackTrace(); // Para fins de debug
    }
    String nome = strData.substring(strData.indexOf('*')+1, strData.indexOf('#'));
    String cat  = strData.substring(strData.indexOf('#')+1, strData.indexOf('%'));
    atletas.put((String)numeros.elementAt(i), new Atleta(nome, Integer.parseInt(cat)));
    //System.out.println(nome + " " + cat); // Para fins de debug
     }
     return true;
}


public Atleta removeAtleta(String numero)
{
     return (Atleta)atletas.remove(numero);
}


public void addAtleta(String key, Atleta atl)
{
     atletas.put(key, atl);
}


public void saveBase()
{
     openRecStore();
  try
  {
    ByteArrayOutputStream strmBytes = new ByteArrayOutputStream();
    DataOutputStream strmDataType = new DataOutputStream(strmBytes);
    byte[] record;
    for(int i=0; i<numeros.size(); i++)
    {
     String StrTempNum = (String)numeros.elementAt(i);
     Atleta atl = (Atleta) atletas.get(StrTempNum);
     strmDataType.writeUTF(StrTempNum);
      strmDataType.writeUTF(atl.getNome());
      strmDataType.writeInt(atl.getCat());
      strmDataType.writeUTF(atl.getTempo());
      strmDataType.writeInt(atl.getColGeral());
      strmDataType.writeInt(atl.getColCategoria());
      // Limpa qualquer dado buferizado
      strmDataType.flush();
      // Pega os dados do stream de bytes e escreve o registro
      record = strmBytes.toByteArray();
      rs.addRecord(record, 0, record.length);
      strmBytes.reset();
    }
    strmBytes.close();
    strmDataType.close();
    numeros.removeAllElements();
    atletas.clear();
  }
     catch (Exception e)
     {
       e.printStackTrace(); // Para fins de debug
  }
```

```
      closeRecStore();
      Mensagem.showMessage("Mensagem...", "Salvamento de dados realizado com sucesso!");
}


public void restoreBase()
{
      openRecStore();
  try
  {
    byte[] recData = new byte[100];
    ByteArrayInputStream strmBytes = new ByteArrayInputStream(recData);
    DataInputStream strmDataType = new DataInputStream(strmBytes);
    for(int i=1; i<=rs.getNumRecords(); i++)
    {
     rs.getRecord(i, recData, 0);
     String StrTempNum = strmDataType.readUTF();
     numeros.addElement(StrTempNum);
     atletas.put(StrTempNum, new Atleta(strmDataType.readUTF(), strmDataType.readInt(),
              strmDataType.readUTF(), strmDataType.readInt(), strmDataType.readInt()));
      strmBytes.reset();
    }
    strmBytes.close();
    strmDataType.close();
  }
      catch (Exception e)
      {
        e.printStackTrace(); // Para fins de debug
      }
      closeRecStore();
      Mensagem.showMessage("Mensagem...", "Recuperação de dados realizada com sucesso!");
}


private void openRecStore()
{
  try
  {
    // Cria o record store se ele não existir
    rs = RecordStore.openRecordStore(REC_STORE, true);
  }
  catch (Exception e)
  {
    e.printStackTrace(); // Para fins de debug
  }
}


private void closeRecStore()
{
  try
  {
    rs.closeRecordStore();
  }
  catch (Exception e)
  {
    e.printStackTrace(); // Para fins de debug
  }
}


public void deleteRecStore()
{
  if (RecordStore.listRecordStores() != null)
  {
    try
    {
      RecordStore.deleteRecordStore(REC_STORE);
    }
    catch (Exception e)
    {
     //e.printStackTrace(); // Para fins de debug
    }
  }
}
```

```
  public void showDataBase() // Esta função é somente para fins de debug
  {
        for(int i=0; i<numeros.size(); i++)
        {
          String StrTempNum = (String)numeros.elementAt(i);
       Atleta atl = (Atleta) atletas.get(StrTempNum);
       System.out.println("Atleta nº:" + StrTempNum + "\t" + atl.getNome() + "\t"
       + atl.getNomeCat(atl.getCat()) + "\tTempo: " + atl.getTempo()
       + "\tColocação Geral: " + atl.getColGeral() + "\tColocação na Categoria: "
       + atl.getColCategoria());
        }
  }


}
```

Conexao.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       Conexao.java
*       ----------------
*                       Classe que implementa o método de importação e exportação para a
*               base de dados hospedada no servidor, é utilizada pela base de dados.
*/


package TriathlonApp;


import javax.microedition.io.*;
import java.io.*;


public class Conexao
{
  private String url;


  public void Conexao()
  {
  }


  public void setURL(String url)
  {
        this.url = url;
  }


  public String getsetData(String quote) throws IOException
  {
        String str = null;
    HttpConnection http = null;
    InputStream iStrm = null;

    try
    {
      // Cria a conexão
      http = (HttpConnection) Connector.open(url + quote);
      // Envia a solicitação
      http.setRequestMethod(HttpConnection.GET);
      // Envia a informação de cabeçalho (opcional)
      http.setRequestProperty("User-Agent","Profile/MIDP-1.0 Configuration/CLDC-1.0");
      // Resposta do servidor
      if (http.getResponseCode() == HttpConnection.HTTP_OK)
      {
        // Pega os dados
        iStrm = http.openInputStream();
        int length = (int) http.getLength();
```

```
        if (length != -1)
        {
          // Lê tudo de uma vez
          byte serverData[] = new byte[length];
          iStrm.read(serverData);
          str = new String(serverData);
        }
        else // Se o comprimento não está disponível
        {
          ByteArrayOutputStream bStrm = new ByteArrayOutputStream();
          // Lê um caracter por vez
          int ch;
          while ((ch = iStrm.read()) != -1)
            bStrm.write(ch);
          str = new String(bStrm.toByteArray());
          bStrm.close();
        }
      }
    }
    finally
    {
      // Fechando a conexão
      if (iStrm != null)
      iStrm.close();
      if (http != null)
      http.close();
    }
    return str;
  }

}
```

Mensagem.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       Mensagem.java
*       ----------------
*                       Classe que exibe uma mensagem de texto para o usuário.
*/


package TriathlonApp;


import javax.microedition.lcdui.*;



public class Mensagem
{

  public Mensagem()
  {}

  public static void showMessage(String header, String msg)
  {
    Alert alert = new Alert(header);
    alert.setTimeout(Alert.FOREVER);
    alert.setString(msg);
    Display.getDisplay(TriathlonAppMain.instance).setCurrent(alert);
  }

}
```

ConfirmacaoParar.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       ConfirmacaoParar.java
*       ------------------------
*                       Classe que exibe a confirmação de parada do cronômetro.
*/


package TriathlonApp;


import javax.microedition.lcdui.*;



public class ConfirmacaoParar extends Form implements CommandListener
{
  private StringItem si;
  private Command cmYes;
  private Command cmNo;
  private Cronometro crm;
  private Display display;

  public ConfirmacaoParar(Cronometro previous)
  {
        super("Atenção!");
        si = new StringItem("Tem certeza que deseja parar o cronômetro?", null);
        cmYes = new Command("Sim", Command.OK, 1);
        cmNo = new Command("Não", Command.CANCEL, 1);
        append(si);
    addCommand(cmYes);
    addCommand(cmNo);
    setCommandListener(this);
    crm = previous;
    display = Display.getDisplay(TriathlonAppMain.instance);
    display.setCurrent(this);
  }


  public void commandAction(Command c, Displayable s)
  {
        if(c == cmYes)
        {
      crm.processStop();
          display.setCurrent(crm);
    }
        else if(c == cmNo)
          display.setCurrent(crm);
  }


}
```

ConfirmacaoSair.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       ConfirmacaoSair.java
*       -----------------------
*                       Classe que exibe a confirmação de saída do aplicativo.
*/


package TriathlonApp;
```

```
import javax.microedition.lcdui.*;


public class ConfirmacaoSair extends Form implements CommandListener
{
  private StringItem si;
  private Command cmYes;
  private Command cmNo;
  private MenuPrincipal menu;
  private Display display;


  public ConfirmacaoSair(MenuPrincipal previous)
  {
       super("Atenção!");
       si = new StringItem("Tem certeza que deseja sair?"
                   + "\nDados não exportados ou não salvos serão perdidos.", null);
       cmYes = new Command("Sim", Command.OK, 1);
       cmNo = new Command("Não", Command.CANCEL, 1);
       append(si);
    addCommand(cmYes);
    addCommand(cmNo);
    setCommandListener(this);
    menu = previous;
    display = Display.getDisplay(TriathlonAppMain.instance);
    display.setCurrent(this);
  }


  public void commandAction(Command c, Displayable s)
  {
       if(c == cmYes)
         TriathlonAppMain.quitApp();
       else if(c == cmNo)
         display.setCurrent(menu);
  }


}
```

Sobre.java

```
/* "Sistema para gerenciamento e administração de competições de triathlon"
*
*       Alunos: Denis Hauffe e Fernando Laudares
*
*       Sobre.java
*       ----------------------
*                     Classe que informa apenas sobre o que faz o aplicativo e quais
*              são seus autores. É subclasse de Mensagem.
*/

package TriathlonApp;


import javax.microedition.lcdui.*;



public class Sobre extends Mensagem
{
  private static String info =
    "TriatlonApp é um Midlet desenvolvido como parte do trabalho de conclusão no curso de "
    + "Bacharelado em Ciências da Computação da Universidade Federal de Santa Catarina.\n\n"
    + "Alunos:\n"
    + "Denis Hauffe e \n"
    + "Fernando Laudares";


  public Sobre() {}
```

```
  public static void showAbout()
  {
       Display display = Display.getDisplay(TriathlonAppMain.instance);
    Alert alert = new Alert("Sobre TriatlonApp");
    alert.setTimeout(Alert.FOREVER);
    alert.setString(info);

    if (display.numColors() > 2)
    {
      String icon = (display.isColor()) ?
                "/icons/JavaPowered-8.png" : "/icons/JavaPowered-2.png";
      try
      {
        Image image = Image.createImage(icon);
           alert.setImage(image);
      }
      catch (java.io.IOException x)
      {
         // Só não adiciona a imagem
      }
    }

    display.setCurrent(alert);
  }

}
```

SISTEMA PARA GERENCIAMENTO E ADMINISTRAÇÃO DE COMPETIÇÕES DE *TRIATHLON*

Denis Nazareno Hauffe[1], Fernando Laudares Camargos[1]

[1]Ciências da Computação, 2004
Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-900
Fone (48) 331-9739, Fax (48) 331-9770
denis@inf.ufsc.br, laudares@inf.ufsc.br

Resumo

Este artigo apresenta um novo sistema de gerenciamento e administração de competições de triathlon de baixo custo e capaz de identificar o atleta no momento em que o mesmo cruza a linha de chegada. O sistema foi desenvolvido utilizando a tecnologia Java 2, Micro Edition (J2ME) e é executável em dispositivos portáteis como aparelhos de telefonia celular e Portable Digital Assistents com suporte a J2ME.

Palavras-chave: *triathlon*, sistema de gerenciamento, J2ME.

Abstract

This article presents a new low cost system for management of triathlon competitions that is capable of identify the athlete in the same moment that he crosses the finish line. The system was developed with the Java 2, Micro Edition (J2ME) technology and is executable in portable devices such as mobile phones and Portable Digital Assistents with J2ME support.

Key-words: *triathlon*, management system, J2ME.

INTRODUÇÃO

O *triathlon* é um esporte criado no final da década de 1970 e composto de três esportes olímpicos: a natação, o ciclismo e o atletismo (mais precisamente o pedestrianismo). Assim como os esportes que o compõe, o *triathlon* é também um esporte dito *de rendimento* – em toda competição de *triathlon* está presente um cronômetro, utilizado para mensurar a performance dos atletas participantes, controlando, assim, a ordem de chegada dos mesmos.

Atualmente, em uma competição que reúne um grande número de atletas, entre amadores e profissionais, a principal dificuldade encontrada pelos organizadores de eventos é como determinar a posição (colocação) alcançada por um atleta ao cruzar a linha de chegada. Relativo à categoria dos profissionais, a resposta seria óbvia: o primeiro homem e a primeira mulher a cruzarem a linha de chegada serão os campeões. Mas quando a questão é transferida para o âmbito da categoria dos amadores, como, por exemplo, quem é a campeã da categoria '45-50 anos', a perspectiva é outra. Muitas vezes, a campeã desta categoria chega muitos minutos, ou mesmo horas, atrás dos primeiros colocados gerais da competição, e cruza a linha de chegada sem saber que venceu, e, conseqüentemente, sem ter a oportunidade de comemorar sua vitória.

Existem, tanto a nível nacional como a nível mundial, dois sistemas diferentes utilizados para solucionar esse problema do controle das parciais e, conseqüentemente, (da ordem de chegada)

dos atletas. Um deles é totalmente automatizado e utiliza um *microchip* codificado, que o atleta carrega preso ao tornozelo, como controle. No momento em que o atleta cruza a linha de chegada, um sensor ali colocado o identifica e registra seu tempo final de prova e colocação no sistema. O outro sistema é inteiramente manual. Fiscais de prova utilizam um cronômetro e uma planilha de papel para registrar o tempo e a colocação dos atletas.

O primeiro sistema é o mais eficiente, divulgando o resultado da competição "em tempo real", porém apresenta um alto custo de implementação que o torna inviável para as competições de pequena escala. O segundo sistema apresenta um custo próximo a zero, porém a lentidão no processo de divulgação dos resultados torna a sua utilização justificável apenas quando o orçamento da competição é um grande impeditivo.

O novo sistema desenvolvido e aqui apresentado, chamado *TriathlonApp*, absorveu o que há de melhor nos sistemas existentes, resultando num sistema de baixo custo capaz de divulgar os resultados de forma rápida e eficiente.

J2ME, A PLATAFORMA JAVA PARA PEQUENOS DISPOSITIVOS

Java 2 Plataform, Micro Edition (J2ME) é a versão da plataforma Java para pequenos dispositivos que contém microprocessadores embutidos (ou embarcados), tais como *Personal Digital Assitents* (PDAs), telefones celulares e conversores de TVs à cabo. Foi a linguagem escolhida para implementar o novo sistema porque é a linguagem para dispositivos com processadores embarcados que mais cresceu nos últimos anos: em 2002 haviam pouco mais de 50 milhões de telefones celulares com tecnologia Java embutida; em 2003, somente a fabricante Nokia vendeu mais de 100 milhões de celulares com Java e a estimativa é que por volta de 2007 praticamente 100% dos telefones celulares executem Java (Almeida, 2004). Além disso, é uma linguagem "interpretada" por uma máquina virtual, o que a torna compatível com as mais diversas plataformas com suporte a J2ME. Essa característica proporciona *portabilidade* ao novo sistema, impedindo que o mesmo fique atrelado a uma única marca/modelo de dispositivo e contribuindo, assim, para manter seu custo baixo.

J2ME é formada por um sub-conjunto das *Application Programming Interface* (API) da versão de Java para computadores *desktop* e servidores (*Java 2 Platform, Standard Edition* (J2SE)), mantendo a compatibilidade com o J2SE sempre que possível.

As características técnicas da arquitetura de cada um destes dispositivos inviabiliza a compatibilidade total entre as duas versões da plataforma Java. Além disso, a justificativa de uma versão especifica para pequenos dispositivos ocorre devido ao fato de que tais dispositivos são caracterizados por possuir um baixo poder de processamento, uma pequena quantidade de memória disponível, presença ou ausência de algum nível de conectividade e um *display* com limitações de tamanho e definição.

Analisando essas características em diferentes dispositivos com microprocessadores embutidos é possível observar que, apesar de compartilharem as mesmas características restritivas apresentadas acima, existe uma diferença perceptível entre as arquiteturas de diferentes dispositivos: enquanto um típico aparelho de telefonia celular apresenta um display com uma resolução total de 12.288 pixels (96x128), um PDA possui um display de pelo menos 20.000 pixels (Muchow, 2001). Apesar de não ser uma diferença tão acentuada quando comparados ao display de um *desktop* convencional (786.432 pixels, 1024x768), ela é significativa em se tratando de construir uma interface para aplicações em pequenos dispositivos.

Em virtude dessas diferenças que existem também entre os dispositivos pequenos, os projetistas da *Sun Microsystems*, criadora da plataforma Java, definiram a J2ME como uma coleção de especificações que definem um conjunto de plataformas, sendo cada um delas adequada para um sub-conjunto dos dispositivos destinados aos consumidores que se enquadram nesse escopo (Topley, 2002).

Assim, o sub-conjunto do ambiente completo de programação Java para um dado dispositivo é definido por um ou mais *perfis*, os quais estendem as capacidades básicas de uma *configuração*. A configuração e o perfil (ou perfis) apropriados para um dispositivo dependem tanto das características da sua arquitetura (*hardware*) quanto do mercado ao qual são destinados (Topley, 2002).

A *configuração* esta fortemente ligada à Máquina Virtual Java (*Java Virtual Machine,* JVM), e define os aspectos da linguagem Java e as bibliotecas utilizadas pela JVM para essa configuração particular em razão, principalmente, da disponibilidade de memória, da capacidade do *display*, do grau de conectividade e do poder de processamento (Muchow, 2001).

O *perfil* é visto como uma extensão da *configuração*, e disponibiliza bibliotecas de classes para programação em um dispositivo específico. Foi a forma encontrada para tornar a J2ME ainda mais específica para um dado dispositivo ao mesmo tempo em que proporciona a ela uma maior flexibilidade aos avanços da tecnologia. O papel do *perfil* é definir APIs para os componentes de interface com o usuário, persistência de informações, manipulação de eventos, *timers*, entre outros, específicos para um ou um número menor de dispositivos que utilizam uma (já específica) *configuração*. A figura 1 ilustra a arquitetura genérica de um dispositivo capaz de executar aplicativos escritos em J2ME: começa pelo Sistema Operacional, como base, seguido pela Máquina Virtual Java (dependente da Configuração), pela Configuração e pelo Perfil.

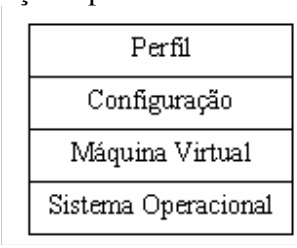

Figura 1 - Arquitetura J2ME genérica

DESENVOLVIMENTO DO SISTEMA

O sistema desenvolvido, chamado TriathlonApp, utiliza a tecnologia de aparelhos de telefonia celular e/ou de PDAs em substituição às planilhas de papel para a obtenção das parciais e um computador servidor, que hospeda a base de dados do sistema e fornece os serviços relacionados à manutenção dessa base de dados.

A conexão entre os dispositivos que realizam o controle das parciais e o servidor agiliza de forma considerável a divulgação da lista com os resultados finais, além de fornecer "instantaneamente" a colocação geral e por categoria de cada atleta que cruzar a linha de chegada.

O sistema foi desenvolvido em duas partes. A primeira delas é caracterizada pela parte da aplicação que é executada no computador servidor: ela inclui o banco de dados onde será armazenada a informação equivalente aos participantes da competição, a interface *web* através da qual esse banco de dados é acessado e também a interface através da qual o aplicativo MIDlet (J2ME), presente nos dispositivos móveis, acessa as informações presentes no banco de dados. A segunda etapa é caracterizada pela parte da aplicação que é executada no(s) dispositivo(s) móvel(eis) (telefone celular e/ou PDA), no formato de um aplicativo MIDlet. Através desse aplicativo é realizado o controle dos resultados da competição de *triathlon*.

Uma das partes críticas do projeto foi definir como a lista com a relação dos atletas participantes da competição seria montada e teria manutenção, além de poder ser "importada" pelo MIDlet. A submissão manual da relação diretamente no dispositivo representava uma tarefa difícil e pouco prática (devido à falta de um teclado completo nos aparelhos), além de caracterizar um trabalho "em duplicata", pois certamente a organização da prova precisaria ter essa mesma lista disponível por inúmeros outros motivos. Dentre as possibilidades disponíveis, optou-se por utilizar um banco de dados que pudesse ser acessado pela Internet através de uma página HTML: assim, o responsável por inserir as informações nessa base de dados (a lista dos participantes) poderia fazê-lo acessando o servidor contendo o banco de dados de qualquer computador com acesso à Internet.

Para manter o sistema funcional com a utilização de *softwares* de livre distribuição e a interação com a linguagem Java, optamos por um conjunto bastante comum e difundido de aplicações: o banco de dados MySQL e a tecnologia *Java Server Pages* (JSP) aliada ao *Apache Tomcat*, o *container* de *servlets* ("aplicações" JSP) usado na implementação de referência da tecnologia JSP.

A interação entre as tecnologias se dá da seguinte forma: inicialmente, o cliente faz uma requisição através de um *browser* de uma página JSP, que nada mais é do que um documento em formato texto que combina *tags* HTML com as *tags* JSP, sendo a página, então, processada pelo servidor. Se for a primeira vez que a página é requisita pelo *browser* ela será transformada em um programa Java (um *servlet*), sendo este compilado, gerando um bytecode que, por sua vez, gera uma página HTML que é enviada de volta ao *browser* do cliente. A partir da segunda vez que esta mesma página for acessada, é verificado se ocorreu alguma modificação na mesma. Se não ocorreu, o *bytecode* armazenado é chamado diretamente, e se ocorreu, o processo de compilação feito pelo *browser* se repete.

Como uma das únicas formas de comunicação dos MIDlets com "o mundo exterior" é através do protocolo HTTP, utilizamos o servidor Apache para hospedar os *servlets* que disponibilizam as informações do banco de dados MySQL, fazendo com que seja possível para o MIDlet acessar (ainda que indiretamente) o banco de dados.

CONCLUSÃO

O objetivo desse trabalho foi desenvolver um sistema de gerenciamento de competições de *triathlon* capaz de identificar cada atleta que cruza a linha de chegada, divulgando, "em tempo real", seu tempo total de prova, sua colocação geral e sua colocação na categoria, sendo, também, capaz de "gerar", poucos minutos após a chegada do último competidor, uma lista contendo nome, colocação, parciais de natação, ciclismo, corrida e tempo total de prova de todos os atletas participantes que completaram a competição (relatório de prova), e que tenha um custo de implementação baixo.

O sistema foi dividido em duas etapas: uma delas rodando no dispositivo, responsável pela captura das parciais durante a competição, e a outra rodando em um servidor, que contém um banco de dados onde é feito o cadastro dos atletas participantes da prova. As duas partes do sistema se comunicam através do protocolo HTTP, através do qual o dispositivo móvel acessa o banco de dados do servidor por intermédio de páginas *web* dinâmicas.

O objetivo principal proposto e alcançado nesse trabalho pode ser resumido na seguinte afirmação: através do novo sistema é possível a identificação do atleta assim que o mesmo cruza a linha de chegada (ou até mesmo antes disso),

tornando possível, no mesmo momento, a divulgação da sua colocação geral, colocação entre os atletas da sua categoria de idade e tempo final de prova.

O novo sistema mantém a característica de baixo custo por ser executável em uma variedade de dispositivos portáteis, tais como aparelhos de telefonia celular e PDAs, bastando para tanto que tais aparelhos possuam suporte à tecnologia J2ME.

REFERÊNCIAS

[1] Almeida, L. B. **Introdução à J2ME e programação MIDP**, Mundo Java, Edição 5, Ano I, 2004, pp.20-27.

[2] Anselmo, F. **Tudo o Que Você Queria Saber Sobre o JSP Quando Utiliza o Servidor TomCat com o Banco MySQL**. Florianópolis: Visual Books, 2002.

[3] Anselmo, F. **Tudo o que você queria saber sobre a JDBC... mas ninguém quis (ou não sabia) responder**. Florianópolis: Visual Books, 2001.

[4] Deitel, H. M. & Deitel, P. J. **Java: Como Programar**, Porto Alegre: Bookman, 4ª edição, 2002.

[5] Hyalen, C.M. **Java 2 Micro Edition: do *pager* ao PDA, com J2ME**, Java Magazine, Edição 1, Ano 1, pp. 32-35.

[6] Martins, F. **Record Store: o pacote javax.microedition.rms**, disponível em www.cpda.com.br, acessado em Junho/2004

[7] Miranda, C. **J2ME no JavaOne 2002: multimídia e jogos no Java Micro Edition**, Java Magazine, Edição 1, Ano 1, pp. 16-19.

[8 Miranda, C. **Conectividade com MIDP: interoperabilidade com serviços externos**, Java Magazine, Edição 6, Ano 1, pp. 22-26.

[9] Miranda, C. **Multimídia no celular: Mobile Media API (MMAPI)**, Java Magazine, Edição 2, Ano 1, pp. 24-26.

[10] Miranda, C. **Dados em J2ME: persistência em dispositivos com RMS e JDBC**, Java Magazine, Edição 3, Ano 1, pp. 20-23.

[11] Miranda, C. **A outra face do J2ME: serviços e novas APIs de intra-estrutura wireless**, Java Magazine, Edição 4, Ano 1, pp. 26-28.

[12] Montenegro, C. **Uso do J2ME em uma aplicação de CRM**, Mundo JAVA, número 2, ano 1, pp. 25-28.

[13] Muchow, J. W. ***Core J2ME: Technology & MIDP***. New Jersey: Prentice Hall PTR, 2001.

[14] Reese, G. *et all*. ***Managing & Using MySQL***. Sebastopol: O'Reilly & Associates, Second Edition, 2002.

[15] Reese, G. *et all*. ***Managing & Using MySQL***. Sebastopol: O'Reilly & Associates, Second Edition, 2002.

[16] Souza, B. **Fazendo wireless acontecer: J2ME e o mercado brasileiro**, in Java Magazine, Edição 3, Ano 1, pp. 12-15.

[17] Topley, K. ***J2ME in a Nutshell***. Sebastopol: O'Reilly & Associates, First Edition, 2002.