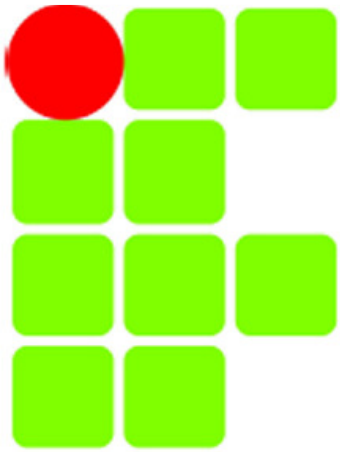

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL DE MINAS GERAIS
Campus Inconfidentes

GILCIMAR DALLÓ

DESENVOLVIMENTO DE UM GERADOR DE AMBIENTES APLICAÇÕES PARA REDES SENSORES NA SOLUÇÃO SUNSPOT

INCONFIDENTES-MG
2013

GILCIMAR DALLÓ

DESENVOLVIMENTO DE UM GERADOR DE AMBIENTES APLICAÇÕES PARA REDES SENSORES NA SOLUÇÃO SUNSPOT

Trabalho de Conclusão de Curso apresentado como pré-requisito de conclusão do curso de Graduação Tecnológica em Redes de Computadores no Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas Gerais – Câmpus Inconfidentes, para obtenção do título de Tecnólogo em Redes de Computadores.

Orientador: Thiago Caproni Tavares

INCONFIDENTES-MG
2013

GILCIMAR DALLÓ

DESENVOLVIMENTO DE UM GERADOR DE AMBIENTES APLICAÇÕES PARA REDES SENSORES NA SOLUÇÃO SUNSPOT

Data de aprovação: 29 de Janeiro de 2013

_____

Orientador: Prof. Msc.Thiago Caproni Tavares
IFSULDEMINAS - Câmpus Inconfidentes

_____

Prof. Luiz Carlos Branquinho Caixeta Ferreira
IFSULDEMINAS - Câmpus Inconfidentes

_____

Prof. Vinícius Ferreira de Souza
IFSULDEMINAS - Câmpus Inconfidentes

Dedico este trabalho de conclusão de curso

aos meus familiares, professores, amigos e colegas que

sempre me apoiaram e principalmente a minha namorada pelo incentivo.

AGRADECIMENTOS

A realização deste trabalho só foi possível graças:

Ao Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas Gerais – Câmpus Inconfidentes.

Aos professores que sempre estiveram a minha disposição para sanar dúvidas e me motivar a criação de ideias inovadoras.

Ao meu orientador que se propôs sem nenhuma restrição a me orientar em busca de um trabalho sólido e que trouxesse bons conhecimentos com as pesquisas.

A todos os colegas e amigos que indiretamente participaram de conversas que me motivou a estudos e pesquisas mais relevantes.

RESUMO

Este Trabalho de Conclusão de Curso tem como principal objetivo estudar Redes Sensores Sem Fio (RSSFs), especificamente trabalhando com cenários no emulador *Solarium* da plataforma *SunSPOT* (*Small Programmable Object Technology*) dos laboratórios da Oracle anteriormente laboratórios da Sun. Para adquirir um bom nível de conhecimento foi necessário estudar os tipos de RSSF e, principalmente, o emulador *Solarium* da plataforma *SunSPOT*. Ao utilizar o emulador *Solarium* foi constatada uma dificuldade na criação de cenários com muitos sensores, pois na ferramenta disponível nativamente no emulador, os sensores são inseridos no cenário um a um, gerando uma grande carga de trabalho para o usuário. Dessa forma, este trabalho visa propor uma ferramenta auxiliar, onde o usuário possa criar um ambiente com muitos sensores informando apenas o número de sensores de cada grupo de aplicação. Após esse processo, a ferramenta gera um arquivo *Extensible Markup Languag* (XML) o qual é importado pelo emulador *Solarium* gerando automaticamente o cenário completo. Contudo, após os testes utilizando a ferramenta criada neste trabalho foi possível identificar um problema que pode atrapalhar na geração de um cenário com muitos sensores, que é o uso de memória computacional, que vai sendo consumida a cada sensor inserido no cenário. Assim pode limitar o número de sensores de acordo com a disponibilidade do *hardware*.

ABSTRACT

This Labor Completion of course has as main objective to study Wireless Sensor Networks (WSN), specifically working with scenarios in the emulator platform Solarium Sunspot (Small Programmable Object Technology) laboratory previously Oracle Sun Labs To acquire a good level of knowledge was necessary to study the types of WSN and, especially, the emulator platform Solarium Sunspot. When using the emulator Solarium was found a difficulty in creating scenarios with many sensors, because the tool available natively on the emulator, sensors are placed on the stage one by one, creating a huge workload for the user. Thus, this paper aims to propose an auxiliary tool, where the user can create an environment with many sensors reporting only the number of sensors of each application group. After this process, the tool generates an Extensible Markup Languag (XML) which is imported by the emulator Solarium automatically generating the complete scenario. However, after testing using the tool created in this study were able to identify a problem that may hinder the generation of a scenario with many sensors, which is the use of computer memory, which is being consumed each sensor inserted into the scenario. Thus it can limit the number of sensors according to the availability of hardware.

LISTAS

LISTA DE FIGURAS

LISTA DE SIGLAS

RSSF ..... Redes de Sensores Sem Fio

MANET ..... Mobile Ad hoc Network

JVM ..... Java Virtual Machine

IDE ..... Integrated Development Environment

LED ..... Light Emitting Diode

RAM ..... Random Access Memory

I/O ..... Input/output

2G ..... Segunda Geração

6G ..... Sexta Geração

JNLP ..... Java Network Launchable Program

SDK ..... Software Development Kit

JRE ..... Java Runtime Environment

USB ..... Universal Serial Bus

JAR ..... Java Archive

XML ..... Extensible Markup Languag

MAC ..... Media Access Control

JAXB ..... Java Architecture for XML Binding

SUMÁRIO

1. INTRODUÇÃO

Os recentes avanços das tecnologias da micro eletrônica e da comunicação sem fio possibilitaram o projeto e desenvolvimento de sensores multifuncionais de baixo custo, baixo consumo de energia e de pequenas dimensões. Tais sensores têm obtido maior capacidade de sensoriamento, processamento e comunicação permitindo a implementação das Redes de Sensores Sem Fio (RSSF) baseada em esforço colaborativo de um grande número de sensores [18].

Uma rede de sensores consiste de um grande número de pequenos sensores que monitoram um ou um conjunto de fenômenos que serão enviados para um usuário final. As redes de sensores podem ser utilizadas em uma gama de aplicações de monitoramento e rastreamento. O grande avanço das aplicações das redes de sensores tem sido possível devido ao desenvolvimento da tecnologia de sensores que estão cada vez menor, economicamente viável e inteligente [19].

Muitos trabalhos estão sendo desenvolvidos utilizando a plataforma *SunSPOT*. Essa plataforma é composta por um hardware de sensoriamento que permite a construção de aplicação para RSSFs. A linguagem utilizada para a programação da plataforma *SunSPOT* é o JAVA que é interpretada por uma máquina virtual chamada *squawk*. Outro ponto positivo da plataforma *SunSPOT* é a provisão de um emulador chamado *Solarium* que executa a máquina virtual *squawk* permitindo a realização de testes prévios antes da instalação na plataforma de sensoriamento. Assim, torna-se mais fácil as implementações das *MIDlets* na camada de aplicação e também um bom desempenho quanto ao consumo de energia, um fator muito

relevante para os pesquisadores já que na maioria das vezes sensores são implantados em locais com pouco acesso a energia.

Uma *MIDlet* é a definição de uma aplicação Java para dispositivos móveis. O termo *MIDlet* é frequentemente utilizado no desenvolvimento de aplicações para celulares. Contudo, esse termo também é utilizado para o contexto da plataforma *SunSPOT*.

Com a fácil implementação de *MIDlets* economiza-se tempo para a realização de testes mais complexos. Dessa forma obtém-se maior confiabilidade nos testes. No entanto existem algumas dificuldades na criação de um ambiente de aplicação como, por exemplo, com uma quantidade elevada de sensores. Isso motivou a criação de uma solução que ajude a melhorar e agilizar a criação dos ambientes com grandes números de sensores de forma rápida e eficaz. Adicionalmente, será possível melhorar o conhecimento de RSSF estudando as aplicações já implantadas e diferenciar os diversos tipos existentes.

O capítulo 2 deste trabalho aborda as fundamentações teóricas utilizadas descrevendo Redes de Sensores Sem Fio a plataforma *SunSPOT* e suas respectivas ferramentas. O capítulo 3 apresenta o principal foco deste trabalho que é trabalhar com cenários em um emulador *Solarium*. O capítulo 4 discorre sobre a implementação de uma ferramenta auxiliar para geração de cenários. Por fim o capítulo 5 apresenta a conclusão deste trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo decorre sobre os estudos realizados sobre RSSF, a plataforma *SunSPOT* e seus componentes.

2.1. REDES DE SENSORES

Redes de Sensores Sem Fio é um tipo especial de rede móvel *ad-hoc* e tem um papel importante na computação ubíqua. Em geral, as RSSFs são formadas por um grande número de dispositivos autônomos chamados nós sensores [1]. Veja um exemplo de topologia na figura 1.

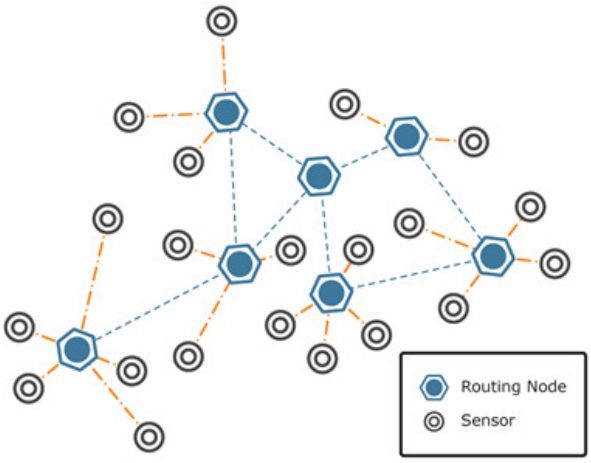

Figura 1: Exemplo de Redes Sensores sem Fio
Fonte: http://www.purelink.ca/en/technologies/related-technologies.php acessado em 20/12/2012

As RSSFs podem ser vistas como um tipo especial de rede móvel ad-hoc (MANET – *Mobile Ad hoc Network*). Em uma rede tradicional, a comunicação entre os elementos computacionais é feita através de estações base de rádio que constituem uma infraestrutura de comunicação como ilustrado na Figura 2. Esse é o caso da Internet. Por outro lado, em uma rede móvel ad-hoc os elementos computacionais trocam dados diretamente entre si, como apresentado na Figura 3. Do ponto de vista de organização, RSSFs e MANETs são idênticas, pois possuem elementos computacionais que se comunicam diretamente entre si através de enlaces de comunicação sem fio. No entanto, as MANETs têm como função básica prover um suporte à comunicação entre esses elementos computacionais, que individualmente, podem estar executando tarefas distintas. Por outro lado, RSSFs tendem a executar uma função colaborativa onde os elementos (sensores) proveem dados, que são processados (ou consumidos) por nós especiais chamados de sorvedouros (*sink nodes*). [2]

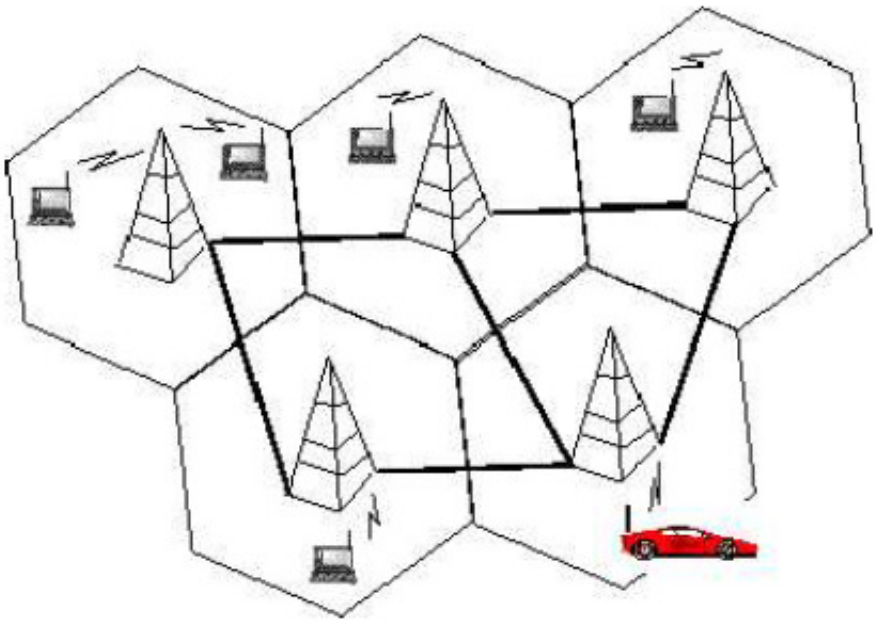

Figura 2: Rede infraestruturada [2].

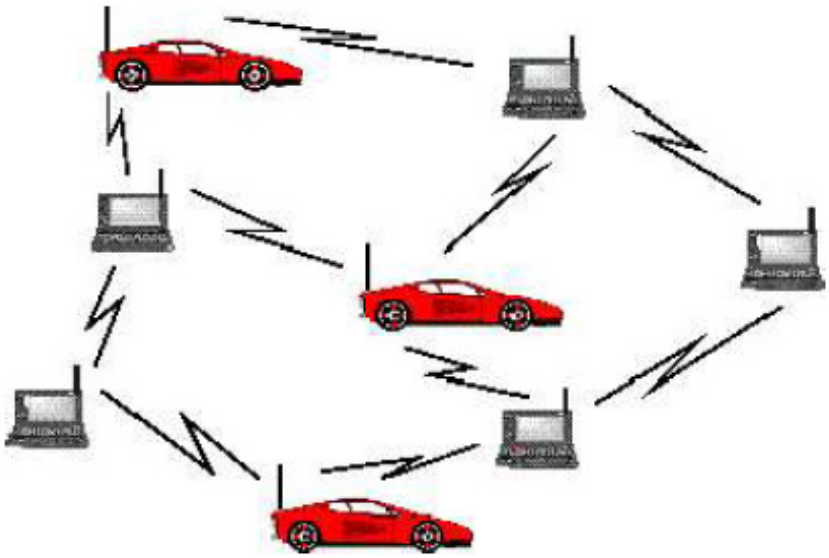

Figura 3: Rede não estruturada [2].

2.2. SUNSPOT

O *Sun Small Programmable Object Technology (SunSPOT)*, desenvolvido pela Sun Microsystem e hoje mantido pela Oracle, é uma tecnologia formada por dispositivos de redes de sensores sem fio que seguem o padrão IEEE 802.15.4, implementando o conjunto de especificações para a comunicação sem fio entre dispositivos eletrônicos denominado *ZigBee* a uma frequência de 2.4 GHz com antena integrada. Esta tecnologia permite que desenvolvedores compilem seus aplicativos Java utilizando a JVM (*Java Virtual Machine*), possibilitando o uso de *IDEs* convencionais para o desenvolvimento dos algoritmos. [4].

Os *SPOTs* têm dimensões pequenas que cabem na palma da mão (vide Figura 4), dispõem de um alto poder computacional, oferecem processamento semelhante aos dos primeiros computadores de meados dos anos 90, utilizam uma pequena máquina virtual Java Micro Edition conhecida como *Squawk* que não necessita de sistema operacional, funcionando diretamente em um núcleo ARM-9 Core de 180 MHz com 512K de memória RAM e 4MB de memória flash. Os mesmos são formados por seis portas analógicas, oito LEDS tricolores, cinco pinos para I/O e quatro pinos de saída de 1998 alta corrente. Esses dispositivos são capazes de perceber o ambiente em que estão inseridos, através de sensores de movimento, temperatura e luz, medidor de aceleração de três eixos, 2G/6G. [3].

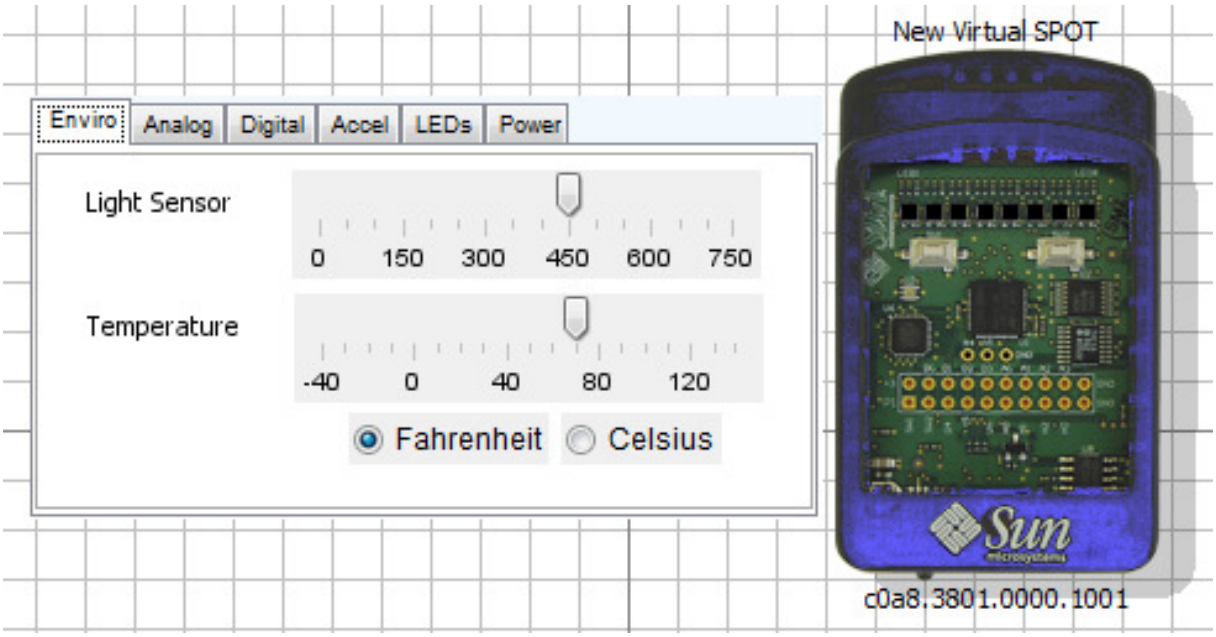

Figura 4: Sensor *SunSPOT*
Fonte: *ScreenShot* do emulador Solarium

Um dos grandes benefícios em utilizar a solução *SunSPOT* é porque ele fornece algumas ferramentas de emulação que ajuda no desenvolvimento de aplicação para o hardware, pois a aplicação, uma vez simulada, basta ser copiada para um sensor real e o funcionamento não requer nenhum reajuste.

Para entender o objetivo deste trabalho é necessário compreender duas ferramentas para o desenvolvimento e emulação das aplicações que utilizam a solução *SunSPOT* que é o *SPOTmanager* e o *Solarium*.

2.3. SPOTMANAGER

É uma ferramenta para manipular o SDK do sensor. A ferramenta é implementada utilizando o *Java Network Launchable Program* (JNLP), isto possibilita que quando a estação de trabalho estiver conectada a internet ela seja atualizada automaticamente para a versão mais recente. [4]

Esta ferramenta provê uma interface agradável que possibilita configurar muitas opções da plataforma *SunSPOT*, por exemplo, qual Software Development Kit (SDKs) será utilizada para desenvolvimento já que existe SDKs diferentes, notícias oficiais disponibilizadas pela empresa Oracle, botões de acionamentos de recursos da plataforma como o botão de execução do emulador *Solarium* e um ótimo console para interações com

comandos e visualização de logs das emulações que são executadas no emulador *Solarium* e outros recursos da plataforma.

2.4. SOLARIUM

Permite gerenciar um grupo de sensores e também gerenciar a aplicação implantada em cada sensor de um cenário. [4].

A Figura 5 apresenta uma implementação simples que descreve uma interface de Gerencialmento nativa do emulador *Solarium*.

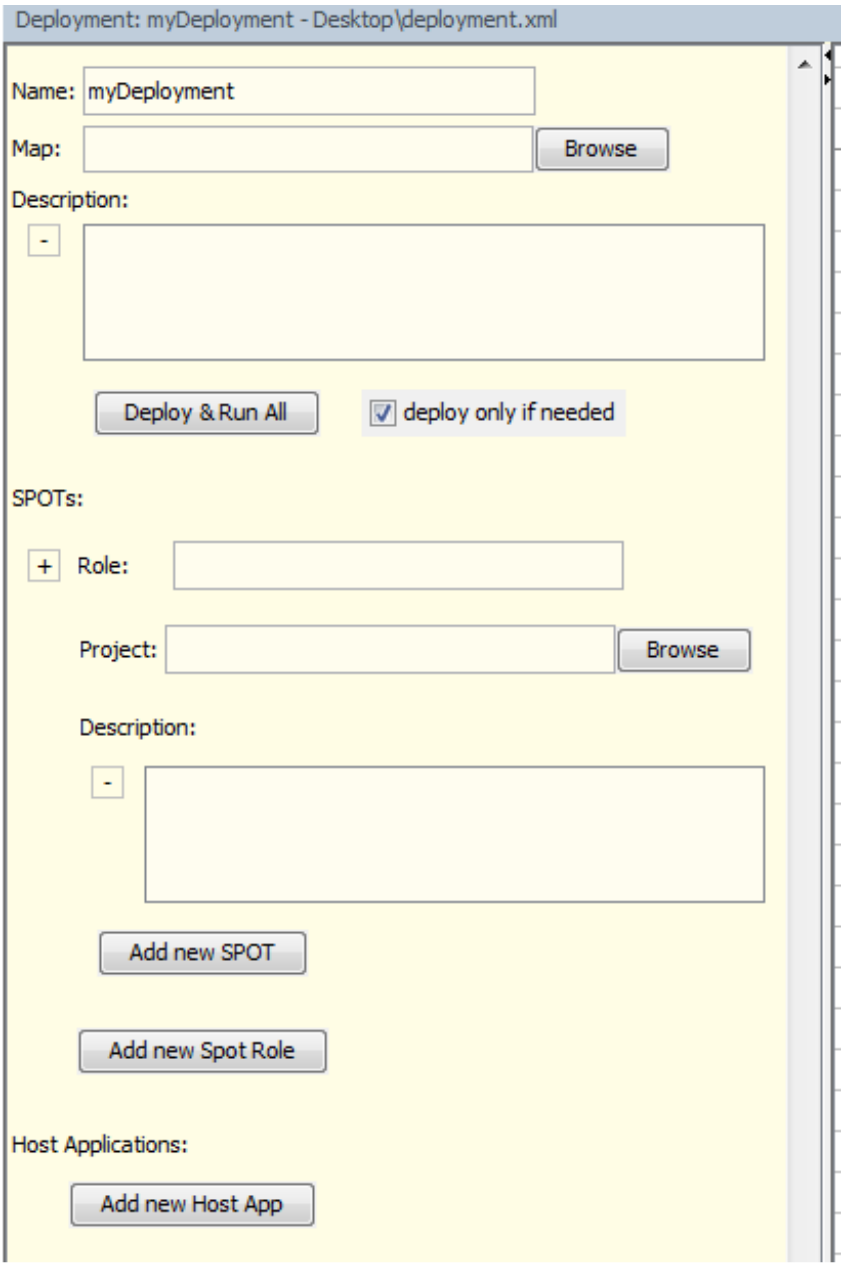

Figura 5: Interface de gerenciamento nativa do emulador solarium.

O emulador *Solarium* fornece as seguintes funcionalidades:

- **Descobrir e exibir sensores:** é possível descobrir sensores que estão conectados via USB no computador e também sensores que se comunicam via rádio[4].
- **Interação com sensores:** pode ser utilizado para carregar e descarregar *softwares* de um sensor, iniciar, pausar, retomar, parar aplicações e consultar o estado atual de um dispositivo, como por exemplo, uso de memória e estatísticas de energia[4].
- **Gestão de uma rede de sensores:** oferece uma visão de implantação especial para facilitar o gerenciamento de uma rede de sensores. Através da ferramenta deve especificar a aplicação que deve ser carregada em cada sensor, e para carregar a aplicação basta clicar em um botão e selecionar a aplicação desejada[4].
- **Emulação de sensores:** é possível emular um sensor virtual para criar e controlar um robô e explorar diversos ambientes físicos[4].

Além disso, o *Solarium* é uma ferramenta baseada na linguagem Java que pode ser usada para gerenciar remotamente uma rede de sensores [4].

Com a utilização do *Solarium* é possível criar, desde pequenas emulações, até emulações complexas envolvendo sensores móveis caracterizando um cenário de robôs. Como o objetivo deste trabalho é lidar com a camada de aplicação dos sensores, serão desconsideradas demonstrações básicas do uso da ferramenta, tais como, especificações de controles básicos. Assim, o trabalho concentra-se no estudo de desenvolvimento de *Midlets* e preparação de cenários utilizando as M*idlets* desenvolvidas [4].

2.5. MANIPULANDO UM SUNSPOT UTILIZANDO SOLARIUM

Para programar a plataforma *SunSPOT* existe algumas vantagens em relação a outros sensores, uma delas é o emulador *Solarium* que possibilita fazer inúmeros testes antes de ter contato com o hardware. Para isso, é importante conhecer alguns conceitos sobre a linguagem Java e sobre a máquina virtual *Squawk,* que é responsável por executar as aplicações dos sensores.

2.6. SQUAWK

É uma máquina virtual Java capaz de rodar em *hardware* sem a necessidade de um sistema operacional.

Os sensores necessitam de uma economia grande de energia. Portanto é necessário a utilização de uma maquina virtual como a *squawk* que não necessita de um sistema operacional.

A máquina virtual *squawk* possui uma arquitetura muito próxima das máquinas virtuais tradicionais da linguagem Java. Dessa forma, os recursos utilizados na programação não sofrem limitações possibilitando uma grande flexibilidade no desenvolvimento. A máquina virtual *Squawk* implementa Java ME.

Java Platform Micro Edition (Java ME) é um ambiente robusto e flexível para aplicações que rodam em celulares e outros dispositivos embarcados.

Em Java ME somente uma aplicação pode ser executada na Máquina Virtual, embora ela possibilite a execução de *threads*. A máquina virtual *Squawk* permite rodar muitas aplicações em um único sensor e usar uma classe especial chamada de *Isolate* para evitar que aplicações interfiram uma nas outras. Cada aplicação roda em uma classe *Isolate* compartilhando o mesmo recurso, porém alguns recursos são únicos como uma conexão de rádio em uma porta específica [4].

Um arquivo JAR pode ser utilizado para iniciar automaticamente quando o ambiente de aplicação é iniciado. Em Java ME é possível que você empacote vários *MIDlets* em um único arquivo JAR. Dessa forma, todos os *MIDlets* devem ser listados em arquivo especial chamado *manifest.mf.*Segue um exemplo do conteúdo de um arquivo *manifest.mf* com mais de uma *MIDlet*:

MIDlet-Name: Example Jar with two MIDlets
MIDlet-Version: 1.0.0
MIDlet-Vendor: Oracle
MIDlet-1: Bounce Demo,, org.sunspotworld.demo.SPOTBounce
MIDlet-2: Air Text Demo, ,org.sunspotworld.demo.AirTextDemo
MicroEdition-Profile: IMP-1.0
MicroEdition-Configuration: CLDC-1.1

2.7. TIPOS DE AMBIENTES EM REDES DE SENSORES

Em redes de sensores podem ser empregadas uma variedade muito grande de ambientes onde podem existir diversos requisitos. Nesta sessão, são identificados alguns aspectos importantes que afetam diretamente uma aplicação em redes de sensores.

A Figura 6 representa uma taxonomia de aplicações. Essa taxonomia define tipos de aplicações de Redes Sensores.

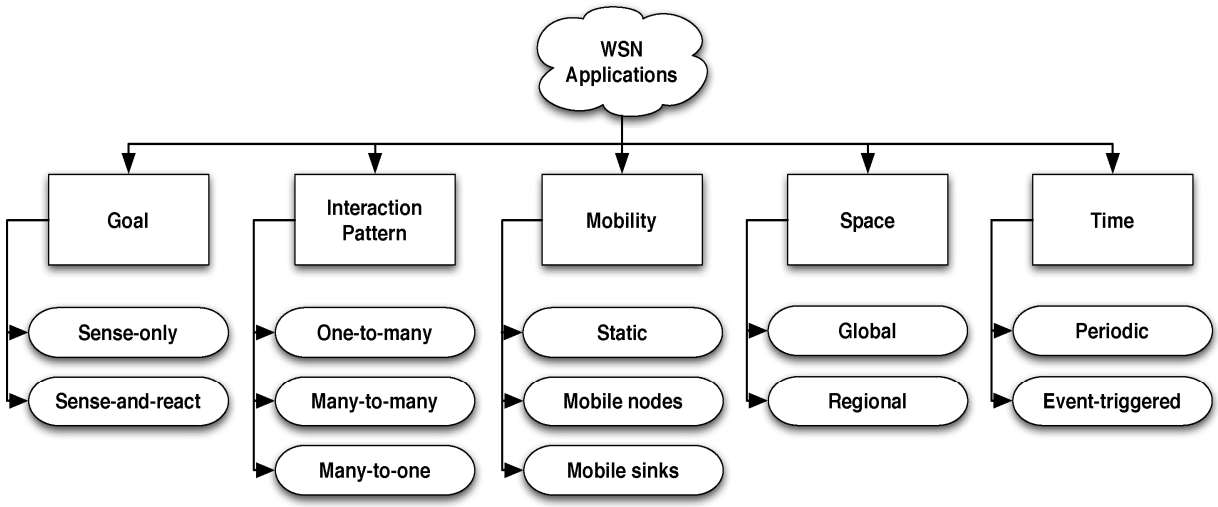

Figura 6: Taxonomia de aplicações em rede sensores [5].

As próximas seções discutem cada um dos elementos da taxonomia apresentada na Figura 6.

2.7.1. *GOAL* (OBJETIVO)

É o momento onde define-se o objetivo da aplicação. Uma aplicação pode ter variados objetivos, por exemplo, coletar dados ambientais.

Os tipos de objetivos podem ser definidos da seguinte forma:

- SENSOR ONLY (SOMENTE SENSOR): Um dos objetivos é a atuação de um sensor de forma que ele só fornecerá os dados. A Figura 7 demonstra um cenário onde ocorre somente um sensoriamento [5].

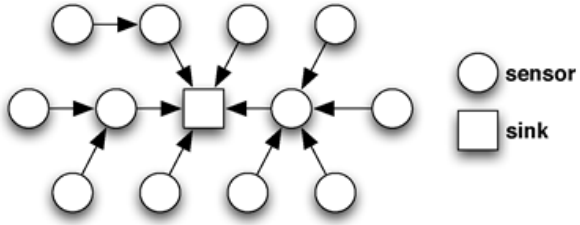

Figura 7: Exemplo de cenário somente com sensor [5].

- SENSOR-AND-REACT (SENSORIAR E REAGIR): Neste caso, existem sensores atuadores que podem reagir a partir de uma determinada condição alcançada [17]. Com este objetivo o cenário pode sofrer modificações. Por exemplo, um cenário onde os sensores atuam como robôs e se movimentam baseados em dados recebidos [5]. Veja a figura 8:

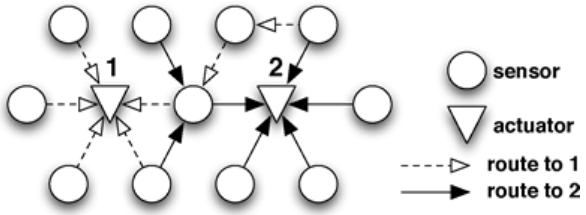

Figura 8: Exemplo de cenário com sensor atuador [5].

2.7.2. *INTERACTION PATTERN* (PADRÃO DE INTERAÇÃO)

Outra distinção fundamental está na forma como os nós da rede interagem, pois isso pode afetar as ações que uma aplicação está programada para executar.

- ONE-TO-MANY (UM PARA MUITOS): É quando um sensor envia uma determinada informação para vários sensores da rede, pode ser considerado desde um *broadcast* até um bloco de sensores com endereços selecionados.
- MANY-TO-MANY (MUITOS PARA MUITOS): É quando todos os sensores enviam dados para todos, os principais cenários que utilizam este padrão de interação são aqueles que possuem robôs.
- MANY-TO-ONE (MUITOS PARA UM): Consiste no tipo mais usado em RSSFs, pois trata de muitos sensores enviando dados a um sensor base (sink node) onde geralmente é encaminhado para nuvem ou armazenado para análise *off-line*.

2.7.3. *MOBILE* (MOBILIDADE)

Redes de sensores sem fio são caracterizadas por topologias altamente dinâmicas, induzidas por flutuações em conectividade típica de propagação sem fio e por ciclo de trabalho padrões necessários para estender a vida útil da rede. No entanto, algumas aplicações introduzem um alto grau de dinamismo, devido à necessidade de apoiar fisicamente dispositivos móveis [5]. Mobilidade pode (ou não) se manifestar de modos diferentes.

- STATIC (ESTÁTICO): Neste modo o sensor não se move de maneira alguma. Atualmente os pesquisadores já passaram por esta fase e focam mais os estudos sensores dinâmicos [5].
- MOBILE NODE (NÓS MÓVEIS): Alguns aplicativos usam os nós móveis ligados a entidades móveis (por exemplo, robôs ou animais) são capazes de se mover de forma autônoma. Um caso típico é o monitoramento da vida selvagem onde os sensores estão ligados a animais, como é o ZebraNet projecto [16].
- MOBILE SINKS (ESTAÇÕES MÓVEIS): É quando um sensor destinado a ser base se move para coletar os dados. O aspecto chave deste caso é que a coleta de dados é realizada de forma oportunista quando o sensor base move próximos aos demais sensores [14].

2.7.4. *SPACE AND TIME* (ESPAÇO E TEMPO)

O processamento distribuído exigido por uma determinada aplicação pode abranger diferentes porções do espaço físico e ser acionado em diferentes instantes no tempo. Estes aspectos são normalmente determinadas pelos fenômenos que estão sendo monitorados. A extensão de processamento distribuído no espaço pode ser:

- GLOBAL: Em aplicações onde o processamento, em princípio, envolve toda a rede, provavelmente porque os fenômenos de interesse abrangem toda a área geográfica onde a RSSF é implantado [5].
- REGIONAL: Em aplicações em que a maioria do processamento ocorre apenas dentro alguma área limitada por um cenário [5].

Já se tratando de tempo, seguem os tópicos relacionados:

- *PERIODIC* (PERIÓDICO): Em aplicações projetadas para processar continuamente dados de sensoriamento, a aplicação executa tarefas periódicas para obter leituras de sensores, coordenadas com outras partes do sistema, e, eventualmente, realiza acionamento conforme necessário [5].
- *EVENT-TRIGGERED* (DISPARAR EVENTO): Em aplicações caracterizadas por duas fases: (i) durante um período programado o nó dispara informações referentes a sua coleta; (ii) quando uma condição é satisfeita com um evento (por exemplo, um valor de sensor sobe acima de um limiar), o sensor começa seu processamento distribuído [5].

3. METODOLOGIA

3.1. CENÁRIOS

É importante destacar que cenários são ambientes onde são implantados sensores para um determinado controle. Cenários são espaços estipulados para que se tenha pontos de sensores realizando atividades das mais diversas possíveis. A Figura 9 apresenta um exemplo de cenário.

Um caso típico é o monitoramento da vida selvagem onde os sensores estão ligados a animais, como é o projeto ZebraNet [16].

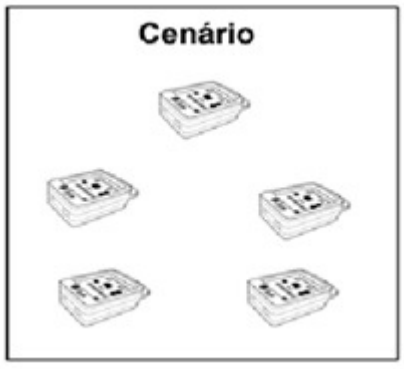

Figura 9: Pequeno cenário estático.

3.2. GERADOR AUTOMÁTICO DE CENÁRIOS

Depois de estudar redes sensores por algum tempo, foi constatado que é uma área de pesquisa muito promissora, e com o passar dos anos vem se consolidando com muitas pesquisas, contudo, ainda existem grandes carências em equipamentos, principalmente em aplicativos que possam emular grandes cenários para praticar os experimentos das pesquisas.

Diante do exposto foi escolhido a solução *SunSPOT* para realizar testes de alguns cenários porque a solução trabalha com a linguagem Java e uma máquina virtual sem a necessidade de um sistema operacional, fato que já interfere num princípio básico em redes de sensores sem fio, que é o consumo de energia. Com de alguns estudos foi constatada uma carência da ferramenta *Solarium* em que, ao tentar criar um cenário com muitos sensores, o tempo gasto é relativamente alto pela necessidade de adicionar nó a nó e relacionar as respectivas aplicações em cada um deles. Isso motivou o desenvolvimento de uma pequena ferramenta auxiliar para gerar os cenários que poderão ser facilmente importado pelo *Solarium*.

Através dos estudos realizados até o momento, foi constado que o *Solarium* nos dá a opção de importação e exportação de cenários para um arquivo o qual é salvo no formato XML. Assim, é possível então a criação de uma ferramenta auxiliar para a geração dos arquivos XML desde que atende os padrões usados pelo *Solarium*.

Mais estudos estão sendo realizados por diversos pesquisadores para o melhor entendimento da geração dos endereçamentos dos sensores o qual é imprescindível em um cenário, pois é através dos endereços físicos que são feitas as comunicações dentro do emulador. Contudo, já foi constatado que a comunicação em camadas abaixo da camada de aplicação não será foco deste trabalho.

Baseado nos levantamentos preliminares, a Figura 10 demonstra a utilização da ferramenta auxiliar para o *Solarium*.

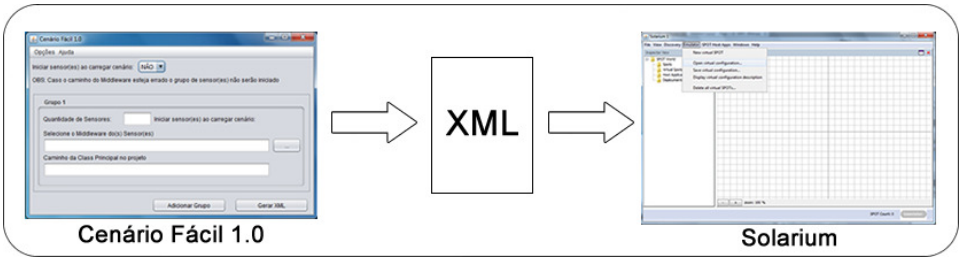

Figura 10: Fluxo de uso do gerador de cenários.

4. IMPLEMENTAÇÃO DA FERRAMENTA CENÁRIO FÁCIL

Constatado a necessidade de criação de uma ferramenta para auxiliar a geração de cenários no emulador *Solarium*, o trabalho foi direcionado ao desenvolvimento da ferramenta em que foi dividido em tópicos que serão apresentados na seguinte sequência: levantamento de requisitos, estudos, desenvolvimento e testes.

4.1. LEVANTAMENTO DE REQUISITOS

Segundo algumas definições que vemos por aí, requisito nada mais é do que uma condição ou capacidade que deve ser alcançada. Simplificando, é algo que um sistema ou componente deve possuir para satisfazer um contrato, padrão ou especificação.

O levantamento de requisitos foi feito baseado nos estudos realizados anteriormente e para facilitar o desenvolvimento foi simulado um cenário simples para verificar as dificuldades na montagem dos cenários.

4.1.1. REQUISITOS FUNCIONAIS

O *SPOTmanager* já estava instalado em um sistema operacional Linux e seus componentes estavam todos em perfeito funcionamento, assim desconsideraremos neste

trabalho explicações sobre o processo de instalação das ferramentas básicas do sensor *SunSPOT* que pode ser detalhada no manual de instalação da plataforma.

Iniciou-se então o emulador *Solarium* utilizando o atalho do *SPOTmanager* e começou o processo de montagem de um cenário teste. Foi escolhido uma aplicação demonstrativa do próprio emulador chamada *LEDSampleCode*, isso auxiliou o levantamento do que seria necessário na ferramenta auxiliar de geração de cenário e quais seriam as necessidades das ferramentas, presumindo que, todo sensor precisa de uma aplicação para que o *hardware* funcione, conseguiu-se levantar o seguinte requisito:

- o usuário deverá selecionar se a aplicação que será carregada no sensor será iniciada automaticamente ao carregar o cenário no emulador;
- poderá existir aplicação diferente em um cenário;
- quantos sensores de cada tipo de aplicação terá o cenário;
- qual o caminho do arquivo *build* de cada aplicação;
- como será a disposição dos sensores no cenário;
- como gerar o endereço físico (MAC) de cada sensor;
- deverá ser gerado um arquivo XML nos padrões de leitura do emulador *Solarium*.

4.1.2. PORTABILIDADE

Ao se pensar em desenvolver uma ferramenta complementar, uma das coisas a se questionar é como será a portabilidade da ferramenta. Isso não foi encarado como um problema, pois como o emulador já trabalha com uma máquina virtual Java decidiu-se então utilizar a linguagem Java para o desenvolvimento, mantendo o foco em uma única linguagem e garantindo a portabilidade para vários sistemas operacionais. Não se pode esquecer que na maioria dos sistemas operacionais, inclusive de equipamentos portáteis, é possível instalar uma maquina virtual Java.

Contudo, a linguagem Java vem se difundindo e a cada versão lançada é implementada novas funcionalidades de desenvolvimento. Com isso, a de se tomar cuidado em usar funcionalidades presentes em versões mais recentes para evitar transtornos à um usuário que terá que atualizar sua maquina virtual mesmo sabendo que a atualização para uma nova versão estável é uma boa prática sugerida pelos desenvolvedores. Dessa forma será

tomada como base o SDK 6.0 para o desenvolvimento sendo necessário o usuário ter instalado o *Java Runtime Environment* (JRE) 6.0 para o bom funcionamento da ferramenta auxiliar.

4.1.3. SEGURANÇA

Por se tratar de uma ferramenta auxiliar não há a necessidade da criação de um mecanismo de segurança como uma tela de autenticação, pois o sistema gerará um arquivo XML seguindo as conformidades exigidas para a leitura no emulador *Solarium* e também os dados contidos no XML são de livres acessos não havendo necessidade de proteção.

4.1.4. USABILIDADE

O sistema é considerado simples sem a necessidade de manuais de utilização, mesmo assim, no próprio sistema deverá conter uma opção de ajuda para que qualquer usuário manipule a ferramenta sem dificuldades e de forma rápida, o que atenderá as expectativas dos utilizadores e diminuirá os trabalhos ao montar um cenário complexo.

4.2. ESTUDOS

4.2.1. SOLARIUM

Em estudos anteriores do emulador *Solarium* foi abordado um aspecto mais global a fim de conhecer todas as suas funcionalidades e aprender sobre a tecnologia *SunSpot*. Neste tópico, será considerado um estudo mais focado ao objetivo da ferramenta auxiliar que é como o emulador trabalha com o arquivo XML de um cenário e as propriedades contidas nele.

A primeira atitude foi montar cenários diferentes com a utilização do emulador e em cada cenário utilizou-se especificidades diferentes e foi exportado um XML com as configurações. Diante disso, foi possível compreender o funcionamento da estrutura do XML. Após levantamento de vários elementos utilizados no arquivo XML, foi realizada uma verificação no manual da plataforma *SunSpot* na tentativa de confrontar as informações para garantir o bom uso dos elementos encontrados no XML, porém não foi encontrado nenhuma

especificidade a se comparar pelo fato de que o manual é de utilização da solução e não um manual de programação ou similar.

Foram detectados os elementos principais do XML que garante a importação de um cenário para o emulador. Seguem abaixo alguns deles:

- ***Virtual-config:*** elemento principal de um cenário onde contém as propriedades responsáveis por informar se as *midlets* serão iniciadas ao carregar o cenário, se serão utilizados endereços físicos (MAC) informados pelo usuário ou se o emulador irá gerar o MAC de cada sensor automaticamente. Foi ai a descoberta que o emulador já gerava o MAC não havendo a necessidade de criação de códigos a fim de gerar os MACs.
- ***Virtual-spot:*** elemento que define um sensor. A princípio, sem nenhuma propriedade relevante.
- ***Build:*** elemento responsável pela aplicação que será carregada no sensor, como o caminho do arquivo *build.xml* que contém informações da aplicação.
- ***Midlet:*** elemento responsável pelas propriedades da *midlet* carregada no elemento *build,* como o nome do pacote contendo a classe principal da aplicação.
- ***Position:*** elemento responsável pelas propriedades de posicionamento do sensor no cenário, como as coordenadas x e y.

4.2.2. JAVA

Um estudo sobre quais técnicas utilizar foi necessário a fim de garantir o bom uso da linguagem evitando reescrever códigos já escritos, uma vez que um dos principais objetivos da linguagem é a reutilização de códigos.

Após várias leituras em busca de um melhor recurso para utilização de uma biblioteca que auxiliasse na criação dos arquivos XML, foi destacado a biblioteca *Java Architecture for XML Binding* (JAXB) por ser nativa da SDK do Java, com isso, não há necessidade de inclusão de bibliotecas de terceiros no código. Para uma boa utilização do JAXB, para cada elemento do XML que forma o cenário deverá ser criado um objeto

facilitando a manipulação de propriedades dos elementos e também tornando flexíveis as futuras inclusões em uma próxima versão da ferramenta.

O código deverá ser bem comentado utilizando técnicas do *plugin* Javadoc a fim de gerar uma documentação detalhada das classes, métodos e atributos criados facilitando a reutilização dos códigos criados e até mesmo a continuação do projeto para uma nova versão abrangendo novos recursos da tecnologia *SunSpot*.

4.3. DESENVOLVIMENTO

4.3.1. LAYOUT

Após os principais requisitos serem levantados, o projeto foi nomeado para que se tornasse um marca fácil de ser mencionado. Surge então, o nome Cenário Fácil 1.0 e depois seguimos para o desenvolvimento pensando sempre em uma ferramenta auxiliar simples e objetiva. A ideia foi criar, primeiramente, o layout da aplicação de forma que atendesse as necessidades levantadas. Veja na Figura 11.

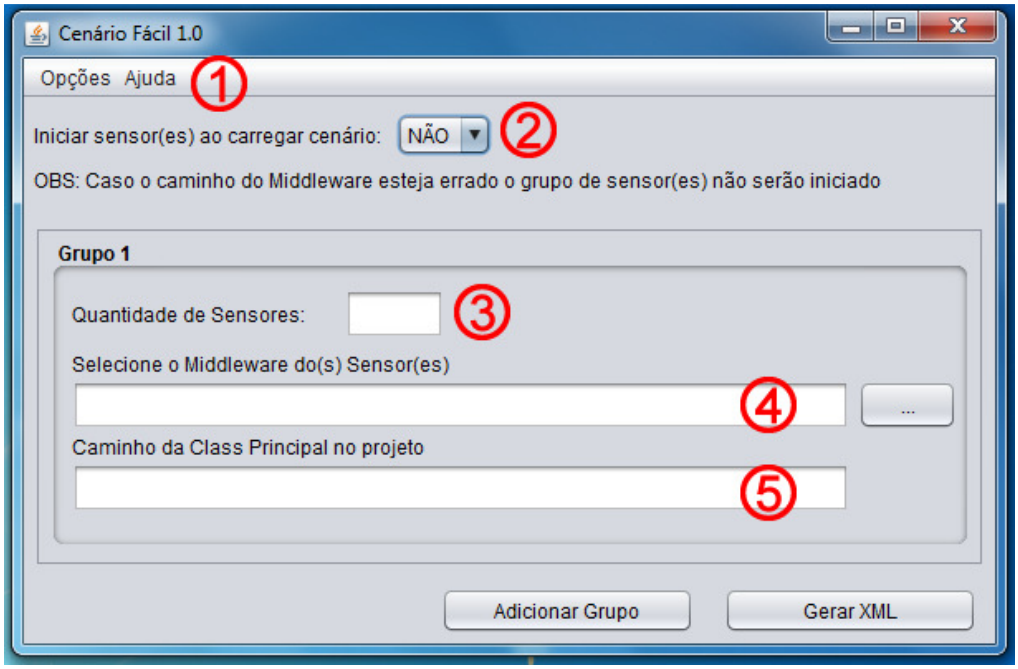

Figura 11: Layout da Ferramenta Auxiliar Cenário Fácil 1.0.

A Figura 11 demonstra a ferramenta Cenário Fácil 1.0, no item 1 esta o menu principal da aplicação contendo as opções “Adicionar um grupo de sensor”, “Gerar XML”, “Encerrar Aplicativo”, “Sobre o *SunSpot*” e “Sobre o Cenário Fácil”. No item 2 é onde o usuário pode optar se as aplicações carregadas nos sensores serão iniciadas automaticamente

ou não. Caso a opção seja não, a aplicação de cada sensor aguardará um comando manual do usuário no emulador para que a aplicação do sensor entre em atividade, casos específicos assim podem acontecer com sensores bases. No item 3, identifica onde o usuário informa o número de sensores que será inserido no cenário com a aplicação. No item 4, o usuário utilizará para selecionar o arquivo *build.xml* que contém as informações necessárias para o carregamento da aplicação para dentro do sensor. No item 5, é o caminho do pacote do projeto até o objeto principal da aplicação e será preenchido automaticamente pelo sistema quando o usuário executar o item 4. A intenção em deixar o campo liberado é para que o usuário visualize se não houve nenhum erro ao compilar sua aplicação antes que ela seja iniciada em um sensor.

Caso o usuário deseja ter mais de uma aplicação em seu cenário, basta clicar na opção "Adicionar Grupo" e será adicionado, dinamicamente, mais um grupo para trabalhar com uma nova aplicação, isso gerará a flexibilidade para montar um cenário com quantas aplicações forem necessárias. O limite de aplicação é indefinido podendo ser limitado pelo cenário proposto por pesquisadores.

4.3.2. CODIFICAÇÃO

Foi utilizado para o desenvolvimento a IDE Netbeans 7.2 para a codificação. Essa IDE foi escolhida pela prática já apresentada pelo desenvolvedor utilizando a mesma, assim não foi considerado qualquer vantagem ou desvantagem comparada a outras IDEs.

A codificação da ferramenta Cenário Fácil 1.0 foi dividida em oito objetos (classes) que estão documentados no Apêndice I deste trabalho.

Apresentamos então os objetos criados neste projeto:

- **Principal**: objeto estendido da biblioteca *Swing,* nativa do Java, sendo responsável pela criação da parte visual do sistema Cenário Fácil 1.0. O objeto possui diversos métodos e atributos que garante o trabalho dinâmico do *layout* desenvolvido. Foi trabalhada a prática de reutilização de códigos sendo utilizados objetos prontos fornecidos pela IDE, como botões e *textField* evitando escrever todo o código do objeto principal,

porém, as regras de negócio do objeto foi toda elaborada no momento do desenvolvimento.

- **CenarioFacil**: objeto responsável por montar o cenário com diversas aplicações utilizando objeto *colletion,* nativo do Java, também responsável por gerar o arquivo XML final que será importado pelo emulador *Solarium*.
- **VirtualConfig**: objeto responsável por criar o elemento *root* do XML, nele é possível citar as propriedades da configuração principal do cenário e, dentro de sua estrutura, carregar os elementos necessários para a formação do cenário.
- **VirtualSpot**: objeto responsável por criar o elemento do XML onde define as propriedades de cada sensor. Dentro de sua estrutura é carregado os elementos específicos de cada sensor, tais como: caminho, posição e outros.
- **Build**: objeto responsável por criar o elemento *build* do XML que é inserido dentro de cada sensor. Este objeto garante o encapsulamento das informações evitando erros de inserção direta ao XML sem os devidos tratamentos.
- **Midlet**: objeto responsável por criar o elemento *midlet* do XML que é inserido dentro de cada sensor. Este objeto garante o encapsulamento das informações evitando erros de inserção direta ao XML sem os devidos tratamentos.
- **Position**: objeto responsável por criar o elemento *position* do XML que é inserido dentro de cada sensor. Este objeto garante o encapsulamento das informações evitando erros de inserção direta ao XML sem os devidos tratamentos.
- **Posicoes**: objeto responsável por calcular o posicionamento dos sensores dentro do simulador *Solarium*. Este objeto, através de cálculos matemáticos, identifica a necessidade de crescimento da área de plotagem e calcula a posição de forma que os sensores sejam inseridos na proporção de um espiral a partir do ponto zero do eixo X e Y do plano. Vale considerar que o posicionamento não segue exatamente um desenho em

espiral, em forma de circunferência, mas tende a crescer de forma homogênea em todos os lados do plano.

4.4. TESTES

Os testes da ferramenta foram realizados constantemente durante o desenvolvimento para evitar grandes falhas ao final. Porém, após a conclusão do projeto foi feito um teste final emulando diversos tipos de cenários estudados em artigos anteriores a fim de comprovar a eficácia da ferramenta.

Foi simulado um cenário contendo dois tipos de aplicações sendo elas aplicações contidas no pacote de demonstração vinda com o emulador *Solarium*. Segue na Figura 12 uma imagem de um cenário gerado com a ferramenta Cenário Fácil 1.0.

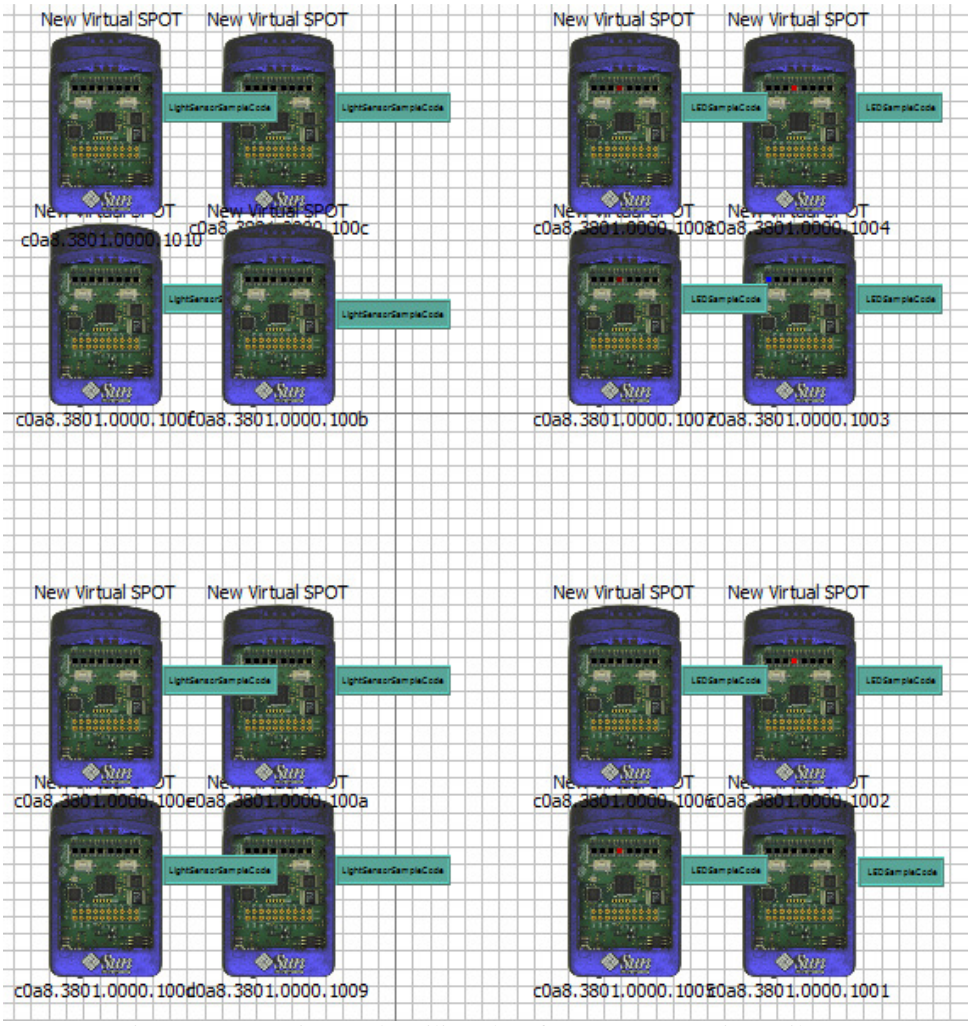

Figura 12: Cenário gerado utilizando a ferramenta Cenário Fácil 1.0.

5. CONCLUSÃO

Foi possível concluir, depois de várias leituras, que realmente Redes Sensores Sem Fio é um foco muito procurado por pesquisadores, pois existem diversos estudos na literatura sobre o assunto. Porém, muitos estudos são implementados em camadas baixas como protocolos de roteamentos, intensidades de sinal entre outros, o qual necessita de aplicações para análise de soluções criadas. Assim, após selecionar o emulador *Solarium* da Tecnologia *SunSPOT* como foco deste trabalho, foi identificado que a ferramenta já possui aplicações de alto nível para os testes de cenários implementados por pesquisadores, no entanto, foi constatado a necessidade de uma ferramenta auxiliar de geração de cenários para ajudar na produtividade dos testes realizados neste emulador.

Na etapa seguinte, o foco do estudo foi todo voltado ao desenvolvimento da ferramenta que foi denominada Cenário Fácil 1.0. O desenvolvimento se voltou à criação de uma ferramenta simples e independente da plataforma *SunSPOT*, de forma a ajudar na criação de cenários complexos a serem emulados na ferramenta Solarium.

A ferramenta Cenário Fácil 1.0, tem como objetivo gerar grandes cenários de forma dinâmica e rápida com apenas alguns cliques, o que não proporcionava diretamente no emulador. Após vários testes, utilizando a ferramenta auxiliar, foi constatado que a ferramenta se torna muito útil na geração de cenários complexos possibilitando emulações com grandes números de sensores e diversas aplicações diferentes com um curto período de tempo na montagem dos cenários. Além do mais, não houve a necessidade de qualquer

alteração no emulador, uma vez que o mesmo possui o serviço de importação de arquivos de configurações do tipo XML, o qual a ferramenta Cenário Fácil 1.0 gera depois de informada as quantidades de sensores e tipos de aplicações de forma simples e rápida.

Algumas dificuldades foram encontradas durante os testes. Ao tentar emular cenários com grandes números de sensores, o custo de uso de memória do computador durante o carregamento da configuração se elevou de forma considerável, deixando muito lento o processo, o que fez concluir que para emulação de um grande cenário necessita-se de um computador com um alto nível de *hardware* para evitar transtornos.

Assim pode-se também destacar que foi levantado mais um ponto de estudo a ser encaminhado aos desenvolvedores da plataforma *SunSPOT:* um melhoramento no consumo de *hardware* quando o emulador *Solarium* está em processo de carregamento de uma configuração salva.

6. BIBLIOGRAFIA

[1] 22° SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 22., 2004, Gramado - Rs. **Arquiteturas para Redes de Sensores Sem Fio.** Gramado - Rs: Sbc, 2004. 52 p.

[2] Loureiro, Antonio A F et al. "Redes de Sensores Sem Fio." *Brasileiro de Redes* 21 (2003) : 179-226.

[3] BRITO, A. V. ; OLIVEIRA,G. S. ; CAETANO, L. J. . **Uma Análise da Implementação ZigBee pela Tecnologia Sun SPOT**. In: XXIX Congresso da Sociedade Brasileira de Computação, 2009, Bento Gonçalves. XXIX Congresso da Sociedade Brasileira de Computação. Porto Alegre : Sociedade Brasileira de Computação, 2009.

[4] INC. MICROSYSTEMS SUN ; ORACLE. SunTM SPOT Programmer's Manual . Sun Labs , 2010. 146 P.

[5] Mottola, L. and Picco, G. P. 2011. Programming wireless sensor networks: Fundamental concepts and state of the art. ACM Comput. Surv. 43, 3, Article 19 (April 2011), 51 pages.

[6] SILVA , M. S. e FRUETT, F. Rede de Sensores sem fio de baixo custo para monitoramento ambiental. XVIII Congresso Brasileiro de Automática , Bonito-MS, 2010. 6 p.

[7] 22° SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 22., 2004, Gramado - Rs. **Middleware Orientado a Serviços para Redes de Sensores sem Fio.** Gramado - Rs: Sbc, 2004. 18 p.

[8] 10° ESCOLA REGIONAL DE COMPUTAÇÃO BAHIA-ALAGOAS-SERGIPE, 10., 2010, Maceió - Al. **MASPOT: Sistema de Agentes Móveis para Sun SPOT.** Maceió - Al: Erbase, 2010. 10 p.

[9] GUIMARÃES, Victor Vinicius Mendoza. **Desenvolvimento de um módulo de monitoramento de consumo de energia para a construção de Smart Appliances utilizando o Sun SPOT.** 2011. 51 f. Trabalho de Conclusão (Bacharel em Engenharia da Computação) - Universidade Federal Do Rio Grande Do Sul, Porto Alegre - Rs, 2011.

[10] G. Wittenburg, K. Terfloth, F. López Villafuerte, T. Naumowicz, H. Ritter, and J. Schiller. Fence monitoring – experimental evaluation of a use case for wireless sensor networks. In Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN), pages 163–178, Delft, The Netherlands, 2007.

[11] W3C (Org.). **Extensible Markup Language (XML).** Disponível em: <http://www.w3.org/XML/>. Acesso em: 18 set. 2012.

[12] JOHNSON, Thienne M. ; MARGALHO, Mauro . Redes de Sensores Sem Fio para Monitoramento Agro-Climatológico na Amazônia. In: Semana Paraense de Informática, 2006, Belém. Semana Paraense de Informática, 2006.

[13] COSTA, F. G. ; BRAUN, T. ; UEYAMA, JÓ ; PESSIN, G. ; OSORIO, F.S. . Arquitetura Baseada Em Veículos Aéreos Não Tripulados E Redes De Sensores Sem Fio Para Aplicações Agrícolas. In: VIII Congresso Brasileiro de Agroinformatica (SBIAgro2011), 2011, Bento Gonçalves, RS. VIII Congresso Brasileiro de Agroinformatica (SBIAgro2011), 2011.

[14] SHAH, R., ROY, S., JAIN, S., AND BRUNETTE, W. 2003. Data MULEs: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Netw. J. 1*, 2–3.

[15] SIERRA, Kathy; BATES, Bert. **Use a Cabeça! Java.** 2. ed. Rio de Janeiro - Rj: Alta Books, 2007. 496 p.

[16] LIU, T. AND MARTONOSI, M. 2003. Impala: A middleware system for managing autonomic, parallel sensor systems. In *Proceedings of the 9th SIGPLAN Symposium on Principles and Practice of Parallel Programming*.

[17] AKYILDIZ, I. F. AND KASIMOGLU, I. H. 2004. Wireless sensor and actor networks: Research challenges. *Ad Hoc Netw. J. 2*, 4.

[18] AKYILDIZ, I.; VURAN, M. C. Wireless sensor networks. New York, NY, USA: John Wiley & Sons, Inc., 2010.

[19] YICK, J.; MUKHERJEE, B.; GHOSAL, D. Wireless sensor network survey. Comput. Netw.,v. 52, n. 12, p. 2292–2330, 2008.

7. APÊNDICE I – DOCUMENTAÇÃO DO CENÁRIO FÁCIL (JAVADOC)

Package cenariofacil

Class Summary	
Build	Classe responsável por criar o elemento build do arquivo XML
CenarioFacil	Classe responsável por montar e gerar o XML do cenário para o emulador SunSPOT
Midlet	Classe responsável por criar o elemento midlet do arquivo XML
Posicoes	Classe responsável por calcular o pisionacionamento ao inserir cada sensor
Position	Classe responsável por criar o elemento position do arquivo XML
Principal	Classe principal do projeto reponsável por gerar a interface do sistema
SobreCenarioFacil	Classe com informações sobre o software Cenário Fácil 1.0
SobreSunspot	Classe com informações sobre a solução SunSpot
VirtualConfig	Classe responsável por criar o elemento principal do XML
VirtualSpot	Classe reponsável por criar o elemento do XML onde define as propriedades de cada sensor (Caminho da aplicação, Posição, Objeto principal da classe)

cenariofacil

Class Build

```
java.lang.Object
  └ cenariofacil.Build
```

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

Classe responsável por criar o elemento build do arquivo XML

Constructor Summary

Build()	Construtor da classe - Não houve necessidade de utilização
Build(java.lang.String file)	Sobrecarga do método construtor

Method Summary

java.lang.String	**getFile**() Método seletor da classe que garante o encapsulamento do atributo file
void	**setFile**(java.lang.String file) Método modificador da classe que garante o encapsulamento do atributo file

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Build

```
public Build()
```

Construtor da classe - Não houve necessidade de utilização

Build

```
public Build(java.lang.String file)
```

Sobrecarga do método construtor

Parameters:

`file` - define o caminho completo da middleware do sensor (String)

Method Detail

getFile

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

Método seletor da classe que garante o encapsulamento do atributo file

Returns:

String - retorna o caminho completo da middleware do sensor gravado dentro do atributo

---

setFile

```
public void setFile(java.lang.String file)
```

Método modificador da classe que garante o encapsulamento do atributo file

Parameters:

`file` - define o caminho completo da middleware do sensor (String)

---

cenariofacil

Class CenarioFacil

```
java.lang.Object
  └ cenariofacil.CenarioFacil
```

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

Classe responsável por montar e gerar o XML do cenário para o emulador SunSPOT

Since:

Versão 1.0 da aplicação

Constructor Summary

CenarioFacil() Método construtor não utilizado

Method Summary

void	**calculaPosicionamento**(int nSensores)
void	**GerarXML**(java.lang.Boolean Init) Método responsável por gerar o XML baseado no ArrayList de sensores
void	**montaCenario**(int nSensores, java.lang.String mid, java.lang.String cClass) Método responsável por montar o cenário

Methods inherited from class java.lang.Object

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

Constructor Detail

CenarioFacil

```
public CenarioFacil()
```

Método construtor não utilizado

Method Detail

calculaPosicionamento

```
public void calculaPosicionamento(int nSensores)
```

montaCenario

```
public void montaCenario(int nSensores,
                         java.lang.String mid,
                         java.lang.String cClass)
```

Método responsável por montar o cenário

Parameters:

`nSensores` - define o número de sensores do cenário (Int)
`mid` - define o endereço do arquivo build.xml do Middleware (String)
`cClass` - define a class principal do projeto (String)

GerarXML

```
public void GerarXML(java.lang.Boolean Init)
              throws javax.xml.bind.JAXBException,
                     java.io.FileNotFoundException
```

Método responsável por gerar o XML baseado no ArrayList de sensores

Parameters:

`Init` - define se o sensores iniciarão as midlets automaticamente (Boolean)

Throws:

`javax.xml.bind.JAXBException` - esta exceção sera lançada a todas exeções JAXB
`java.io.FileNotFoundException` - esta exceção será lançada pelos construtores FileInputStream, FileOutputStream e RandomAccessFile quando um arquivo com o nome do caminho especificado não existe. Também será lançada por esses construtores se o arquivo não existe, mas por alguma razão é inacessível, por exemplo, quando é feita uma tentativa de abrir um arquivo de somente leitura para a escrita.

---

cenariofacil

Class Midlet

```
java.lang.Object
  └ cenariofacil.Midlet
```

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

Classe responsável por criar o elemento midlet do arquivo XML

Constructor Summary

Midlet() Construtor da classe - Não houve necessidade de utilização
Midlet(java.lang.String name) Sobrecarga do método construtor

Method Summary

java.lang.String	**getName**() Método seletor da classe que garante o encapsulamento do atributo name
void	**setName**(java.lang.String name) Método modificador da classe que garante o encapsulamento do atributo name

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Midlet

```
public Midlet()
```

Construtor da classe - Não houve necessidade de utilização

Midlet

```
public Midlet(java.lang.String name)
```

Sobrecarga do método construtor

Parameters:

name - define o nome da classe principal com os referidos pacotes (String)

Method Detail

getName

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

Método seletor da classe que garante o encapsulamento do atributo name

Returns:

String - retorna o caminho completo dos pacotes até a classe principal

---

setName

```
public void setName(java.lang.String name)
```

Método modificador da classe que garante o encapsulamento do atributo name

Parameters:

`name` - define o caminho completo dos pacotes até a classe principal (String)

---

cenariofacil

Class Posicoes

```
java.lang.Object
  └ cenariofacil.Posicoes
```

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

Classe responsável por calcular o pisionacionamento ao inserir cada sensor

Constructor Summary

Posicoes() Construtor da classe - Não houve necessidade de utilização

Method Summary

java.util.ArrayList	**posSensores**(int nSensores, int largura, int altura)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Posicoes

```
public Posicoes()
```

Construtor da classe - Não houve necessidade de utilização

Method Detail

posSensores

```
public java.util.ArrayList posSensores(int nSensores,
                                       int largura,
                                       int altura)
```

Parameters:

`nSensores` - define o número de sensores que deverá ser inserido no simulador SunSPOT (Int)

`largura` - define a largura do sensor no emulador para que nenhum sensor sobreponha o outro na horizontal (Int)

`altura` - define a altura do sensor no emulador para que nenhum sensor sobreponha o outro na

vertical (Int)

Returns:

ArrayList String - Retorna o posicionamento x e y de cada sensor separado por vírgula

---

cenariofacil

Class Position

```
java.lang.Object
  └ cenariofacil.Position
```

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

Classe responsável por criar o elemento position do arquivo XML

Constructor Summary

Position()	Construtor da classe - Não houve necessidade de utilização
Position(int x, int y)	Sobrecarga do construtor - Método que define o posicionamento x e y do elemento position no XML

Method Summary

int	**getX**() Método seletor da classe que garante o encapsulamento do atributo X
int	**getY**() Método seletor da classe que garante o encapsulamento do atributo Y
void	**setX**(int x) Método modificador da classe que garante o encapsulamento do atributo X
void	**setY**(int y) Método modificador da classe que garante o encapsulamento do atributo Y

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Position

```
public Position()
```

Construtor da classe - Não houve necessidade de utilização

Position

```
public Position(int x,
                int y)
```

Sobrecarga do construtor - Método que define o posicionamento x e y do elemento position no XML

Parameters:

`x` - define valor de X (Int)
`y` - define valor de Y (Int)

Method Detail

getX

```
public int getX()
```

Método seletor da classe que garante o encapsulamento do atributo X

Returns:

Int - retorna o posicionamento X do sensor gravado dentro do atributo

---

setX

```
public void setX(int x)
```

Método modificador da classe que garante o encapsulamento do atributo X

Parameters:

`x` - define o valor x a ser modificado no atributo (Int)

---

getY

```
public int getY()
```

Método seletor da classe que garante o encapsulamento do atributo Y

Returns:

Int - retorna o posicionamento Y do sensor gravado dentro do atributo

---

setY

```
public void setY(int y)
```

Método modificador da classe que garante o encapsulamento do atributo Y

Parameters:

`y` - define o valor y a ser modificado no atributo (Int)

---

cenariofacil

Class Principal

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ cenariofacil.Principal
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class Principal
extends javax.swing.JFrame
```

Classe principal do projeto reponsável por gerar a interface do sistema

Since:

Versão 1.0 da aplicação

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes/interfaces inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes/interfaces inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes/interfaces inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes/interfaces inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Principal()
Método construtor que inicializa os componentes

Method Summary

java.lang.String	**arquivo**(java.lang.String caminho) Método responsável por ler o arquivo manifest.mf do projeto e recuperar o caminho da classe principal
static void	**main**(java.lang.String[] args)

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics, getJMenuBar, getLayeredPane, getRootPane, getTransferHandler, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, repaint, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setIconImage, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, setTransferHandler, update

Methods inherited from class java.awt.Frame
addNotify, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getIconImages, getInputContext, getListeners, getLocale, getModalExclusionType, getMostRecentFocusOwner,

getOwnedWindows, getOwner, getOwnerlessWindows, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindows, getWindowStateListeners, hide, isActive, isAlwaysOnTop, isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot, isFocused, isLocationByPlatform, isShowing, pack, paint, postEvent, processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, reshape, setAlwaysOnTop, setBounds, setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot, setIconImages, setLocationByPlatform, setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize, setSize, setVisible, show, toBack, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusTraversalKeys, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBaseline, getBaselineResizeBehavior, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, resize, resize, setBackground, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setMaximumSize, setName, setPreferredSize, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object

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

Methods inherited from interface java.awt.MenuContainer
getFont, postEvent

Constructor Detail

Principal

```
public Principal()
```

Método construtor que inicializa os componentes

Method Detail

arquivo

```
public java.lang.String arquivo(java.lang.String caminho)
```

Método responsável por ler o arquivo manifest.mf do projeto e recuperar o caminho da classe principal

Parameters:

`caminho` - define o caminho do arquivo manifest.mf do projeto (String)

Returns:

String - retorna o pacote com a classe principal do projeto

---

main

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

Parameters:

`args` - os argumentos da linha de comando

---

cenariofacil

Class SobreCenarioFacil

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ cenariofacil.SobreCenarioFacil
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class SobreCenarioFacil
extends javax.swing.JFrame
```

Classe com informações sobre o software Cenário Fácil 1.0

Since:

Versão 1.0 da aplicação

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes/interfaces inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes/interfaces inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes/interfaces inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes/interfaces inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

SobreCenarioFacil()
Creates new form SobreCenarioFacil

Method Summary

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

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics, getJMenuBar, getLayeredPane, getRootPane, getTransferHandler, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, repaint, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setIconImage, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, setTransferHandler, update

Methods inherited from class java.awt.Frame
addNotify, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getIconImages, getInputContext, getListeners, getLocale, getModalExclusionType, getMostRecentFocusOwner, getOwnedWindows, getOwner, getOwnerlessWindows, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindows, getWindowStateListeners, hide, isActive, isAlwaysOnTop, isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot, isFocused, isLocationByPlatform, isShowing, pack, paint, postEvent,

processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, reshape, setAlwaysOnTop, setBounds, setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot, setIconImages, setLocationByPlatform, setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize, setSize, setVisible, show, toBack, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusTraversalKeys, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBaseline, getBaselineResizeBehavior, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, resize, resize, setBackground, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setMaximumSize, setName, setPreferredSize, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object

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

Methods inherited from interface java.awt.MenuContainer

getFont, postEvent

Constructor Detail

SobreCenarioFacil

```
public SobreCenarioFacil()
```

Creates new form SobreCenarioFacil

Method Detail

main

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

Parameters:

`args` - the command line arguments

---

cenariofacil

Class SobreSunspot

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ cenariofacil.SobreSunspot
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class SobreSunspot
extends javax.swing.JFrame
```

Classe com informações sobre a solução SunSpot

Since:

Versão 1.0 da aplicação

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes/interfaces inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes/interfaces inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes/interfaces inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes/interfaces inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

SobreSunspot()
Creates new form SobreSunspot

Method Summary

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

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics, getJMenuBar, getLayeredPane, getRootPane, getTransferHandler, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, repaint, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setIconImage, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, setTransferHandler, update

Methods inherited from class java.awt.Frame
addNotify, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getIconImages, getInputContext, getListeners, getLocale, getModalExclusionType, getMostRecentFocusOwner, getOwnedWindows, getOwner, getOwnerlessWindows, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindows, getWindowStateListeners, hide, isActive, isAlwaysOnTop, isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot, isFocused, isLocationByPlatform, isShowing, pack, paint, postEvent,

processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, reshape, setAlwaysOnTop, setBounds, setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot, setIconImages, setLocationByPlatform, setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize, setSize, setVisible, show, toBack, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusTraversalKeys, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBaseline, getBaselineResizeBehavior, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, resize, resize, setBackground, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setMaximumSize, setName, setPreferredSize, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object

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

Methods inherited from interface java.awt.MenuContainer

getFont, postEvent

Constructor Detail

SobreSunspot

```
public SobreSunspot()
```

Creates new form SobreSunspot

Method Detail

main

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

Parameters:

`args` - the command line arguments

---

cenariofacil

Class VirtualConfig

```
java.lang.Object
  └ cenariofacil.VirtualConfig
```

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

Classe responsável por criar o elemento principal do XML

Constructor Summary

VirtualConfig() Construtor da classe - Não houve necessidade de utilização
VirtualConfig(boolean keep_addresses, boolean run_midlets, java.util.Collection<VirtualSpot> virtualspot) Sobrecarga do construtor - Método que define as opções do elemento principal do sensor no XML

Method Summary

java.util.Collection<VirtualSpot>	**getVirtualspot**() Método seletor da classe que garante o encapsulamento do atributo virtualspot
boolean	**isKeep_addresses**() Método seletor da classe que garante o encapsulamento do atributo keep_addresses
boolean	**isRun_midlets**() Método seletor da classe que garante o encapsulamento do atributo run_midlets
void	**setKeep_addresses**(boolean keep_addresses) Método modificador da classe que garante o encapsulamento do atributo keep_addresses
void	**setRun_midlets**(boolean run_midlets) Método modificador da classe que garante o encapsulamento do atributo run_midlets
void	**setVirtualspot**(java.util.Collection<VirtualSpot> virtualspot) Método modificador da classe que garante o encapsulamento do atributo virtualspot

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

VirtualConfig

```
public VirtualConfig()
```

Construtor da classe - Não houve necessidade de utilização

VirtualConfig

```
public VirtualConfig(boolean keep_addresses,
                     boolean run_midlets,
                     java.util.Collection<VirtualSpot> virtualspot)
```

Sobrecarga do construtor - Método que define as opções do elemento principal do sensor no XML

Parameters:

`keep_addresses` - define se o sensor vai ter ou não um MAC definido pelo usuário (Boolean)
`run_midlets` - define se o sensor inciará automaticamente a midlet (Boolean)
`virtualspot` - define um collection com todos os sensores montados

Method Detail

isKeep_addresses

```
public boolean isKeep_addresses()
```

Método seletor da classe que garante o encapsulamento do atributo keep_addresses

Returns:

Boolean - retorna o valor do atributo keep_addresses

---

setKeep_addresses

```
public void setKeep_addresses(boolean keep_addresses)
```

Método modificador da classe que garante o encapsulamento do atributo keep_addresses

Parameters:

`keep_addresses` - define o valor keep_addresses a ser modificado no atributo (Boolean)

---

isRun_midlets

```
public boolean isRun_midlets()
```

Método seletor da classe que garante o encapsulamento do atributo run_midlets

Returns:

Boolean - retorna o valor do atributo run_midlets

---

setRun_midlets

```
public void setRun_midlets(boolean run_midlets)
```

Método modificador da classe que garante o encapsulamento do atributo run_midlets

Parameters:

`run_midlets` - define o valor run_midlets a ser modificado no atributo (Boolean)

---

getVirtualspot

```
public java.util.Collection<VirtualSpot> getVirtualspot()
```

Método seletor da classe que garante o encapsulamento do atributo virtualspot

Returns:

Collection do tipo VirtualSpot - retorna o valor do atributo virtualspot

---

setVirtualspot

```
public void setVirtualspot(java.util.Collection<VirtualSpot> virtualspot)
```

Método modificador da classe que garante o encapsulamento do atributo virtualspot

Parameters:

`virtualspot` - define o valor virtualspot a ser modificado no atributo (Collection do tipo VirtualSpot)

---

cenariofacil

Class VirtualSpot

```
java.lang.Object
  └ cenariofacil.VirtualSpot
```

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

Classe reponsável por criar o elemento do XML onde define as propriedades de cada sensor (Caminho da aplicação, Posição, Objeto principal da classe)

Constructor Summary

VirtualSpot()	Construtor da classe - Não houve necessidade de utilização
VirtualSpot(Build build, Position position, Midlet midlet)	Sobrecarga do construtor - Método que define as propriedades do sensor no XML

Method Summary

Build	**getBuild**() Método seletor da classe que garante o encapsulamento do atributo build
Midlet	**getMidlet**() Método seletor da classe que garante o encapsulamento do atributo midlet
Position	**getPosition**() Método seletor da classe que garante o encapsulamento do atributo position
void	**setBuild**(Build build) Método modificador da classe que garante o encapsulamento do atributo build
void	**setMidlet**(Midlet midlet) Método modificador da classe que garante o encapsulamento do atributo midlet
void	**setPosition**(Position position) Método modificador da classe que garante o encapsulamento do atributo position

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

VirtualSpot

```
public VirtualSpot()
```

Construtor da classe - Não houve necessidade de utilização

VirtualSpot

```
public VirtualSpot(Build build,
                   Position position,
                   Midlet midlet)
```

Sobrecarga do construtor - Método que define as propriedades do sensor no XML

Parameters:

`build` - define o elemento build do XML
`position` - define o elemento position do XML
`midlet` - define o elemento midlet do XML

Method Detail

getBuild

```
public Build getBuild()
```

Método seletor da classe que garante o encapsulamento do atributo build

Returns:

Objeto build - retorna o elemento build gravado no atributo

---

setBuild

```
public void setBuild(Build build)
```

Método modificador da classe que garante o encapsulamento do atributo build

Parameters:

`build` - define o valor build a ser modificado no atributo

---

getPosition

```
public Position getPosition()
```

Método seletor da classe que garante o encapsulamento do atributo position

Returns:

Objeto position - retorna o elemento position gravado no atributo

---

setPosition

```
public void setPosition(Position position)
```

Método modificador da classe que garante o encapsulamento do atributo position

Parameters:

`position` - define o valor position a ser modificado no atributo

---

getMidlet

```
public Midlet getMidlet()
```

Método seletor da classe que garante o encapsulamento do atributo midlet

Returns:

Ojeto midlet - retorna o elemento midlet gravado no atributo

---

setMidlet

```
public void setMidlet(Midlet midlet)
```

Método modificador da classe que garante o encapsulamento do atributo midlet

Parameters:

`midlet` - define o valor midlet a ser modificado no atributo
