ANDRÉ LUIZ LINS DE AQUINO

BEANWATCHER: UMA FERRAMENTA PARA O DESENVOLVIMENTO DE APLICAÇÕES DE MONITORAÇÃO EM AMBIENTES SEM FIO

Belo Horizonte
28 de Novembro de 2003

ANDRÉ LUIZ LINS DE AQUINO

BEANWATCHER: UMA FERRAMENTA PARA O DESENVOLVIMENTO DE APLICAÇÕES DE MONITORAÇÃO EM AMBIENTES SEM FIO

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte

28 de Novembro de 2003

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

BeanWatcher: Uma Ferramenta para o Desenvolvimento de Aplicações de Monitoração em Ambientes Sem Fio

ANDRÉ LUIZ LINS DE AQUINO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. Dr. Claudionor José Nunes Coelho Junior - Orientador
Departamento de Ciência da Computação – ICEx – UFMG

Prof. Dr. Antônio Alfredo Ferreira Loureiro - Co-Orientador
Departamento de Ciência da Computação – ICEx – UFMG

Prof. Dr. Antônio Otávio Fernandes
Departamento de Ciência da Computação – ICEx – UFMG

Prof. Dr. Diógenes Cecilio da Silva Junior
Departamento de Engenharia Elétrica – UFMG

Belo Horizonte, 28 de Novembro de 2003.

Os homens conseguiriam muito mais coisas
se julgassem menos coisas impossíveis.

Esta dissertação dedico as mulheres de minha vida, Mãe Tama e Jacyara...

Agradecimentos

Em todas as dificuldades e barreiras que encontrei durante meu mestrado, sempre tive diversas pessoas que de uma forma ou de outra me ajudaram a continuar e não desistir no meio do caminho, porém jamais teria conseguido se não tivesse Deus perto de mim que me fez companhia nos momentos solidão, me manteve calmo nos momentos de decisão onde coisas importantes estavam em jogo e me fez capaz o suficiente para poder terminar com louvores esta jornada. Senhor a ti devo minha eterna gratidão.

As pessoas os quais tenho muito amor e carinho que me apoiaram desde criança nos meus primeiros passos, tiveram paciência nas minhas revoltas de adolescência, se mantiveram forte diante a saudade e a distância que nos separa e ainda assim sempre me incentivaram a seguir com meus sonhos. Mãe Tama e pai Luiz, muito obrigado por me colocarem no mundo e serem as pessoas simples e honesta o qual eu amo muito.

A minha irmã que apesar das desavenças quando criança sempre se manteve com pensamento positivo para que as coisas corressem bem comigo, muito obrigado. Ao meu irmão conselheiro Édler pelo apoio e suporte que sempre me deu nos meu momentos mais difíceis, muito obrigado. A minha irmãzinha Larissa que apesar de sua pouca idade me dar esperanças de que o mundo um dia pode ser melhor e que eu possa passar a ela tudo que aprendi na vida.

Apesar de distante, existem pessoas que se mantém fieis a amizade e continuam presentes fazendo parte de nossas vidas. A minha amiga Raquel que se manteve companheira e me ajudou bastante, mesmo distante, em vários momentos de tristeza o qual passei, muito obrigado.

Este precioso trabalho de mestrado não poderia se conceber se eu não tivesse o suporte e o apoio de pessoas muito competentes, inteligentes e acima de tudo amigas. Meu orientador Prof. Claudionor e meu Co-orientador Prof. Loureiro muito obrigado.

Apesar de distante da minha terra que tanto amo e de todas as pessoas os quais me sinto feliz ao estar com elas, consegui conquistar grandes amizades que certamente jamais esquecerei, dentre eles meu amigo Argentino Martin e meu amigo Japonês Nakamura a vocês obrigado pela força e pela amizade que tenho por vocês. Além disso, agradeço a todos os meus amigos daqui de BH pelo companheirismo, alegrias e tristezas compartilhadas.

A Jacyara simplesmente obrigado por sua existência em minha vida.

Resumo

Redes sem fio são um tipo especial de redes de computadores que utilizam ondas de rádio para prover comunicação entre os nós. Para estas redes existem alguns padrões tais como WLANs, WPANs e *Broadband Wireless Access*. Estes padrões permitem que usuários sem fio utilizem sua rede em diferentes cenários como escolas, hospitais e indústrias. Estima-se que em 2001, 16 milhões de pessoas usaram o padrão IEEE 802.11 e que em 2006, este número irá crescer para 60 milhões. Dentro deste contexto existe uma variedade de aplicações, como aplicações militares, ambientais, médicas e industriais, onde para desenvolver estas aplicações normalmente utiliza-se soluções específicas para cada problema, tornando a manutenção e extensão uma difícil tarefa, logo a utilização de uma ferramenta que automatize o esforço do desenvolvedor, generalizando os elementos que compõem estas aplicações, é de grande utilidade.

Este trabalho propõe uma ferramenta baseada em componentes, chamada BeanWatcher, que permite o desenvolvimento de aplicações de monitoração em ambientes sem fio que executam em um dispositivo portátil móvel, podendo agir tanto como monitor de um dado remoto como atuador caso a aplicação necessite interferir no meio monitorado. Para conceber esta ferramenta foi utilizado um modelo do componentes que padroniza os elementos monitorados, tornando as aplicações mais simples e portáteis, pois através do BeanWatcher é possível a geração de aplicações em diferentes ambientes como Java, J2me e C++.

Como estudo de caso, desenvolvemos aplicações para redes de sensores. Tais redes possuem capacidade de sensoriamento e processamento distribuído. Fatores relacionados com as características da rede, tipos e configurações dos sensores influenciam diretamente no desenvolvimento das aplicações para estas redes. Com isso, desenvolvemos através do BeanWatcher aplicações para monitoramento médico, permitindo que os dados vitais dos pacientes sejam visto no Palm do médico. Aplicação para controle industrial, onde o operário pode monitorar a temperatura dos equipamentos e controlá-los remotamente. Aplicação agropecuária que permite que o agricultor monitore dados ambientais numa determinada região da plantação. Estas aplicações foram desenvolvidas com o objetivo de mostrar a viabilidade e eficiência na utilização do BeanWatcher.

Abstract

A Wireless Network is a special type of network that uses radio waves to communicate between nodes. Recent standard approvals for WLANs, WPANs, and Broadband Wireless Access allowed users to wirelessly extend their networks in different scenarios such as schools, enterprises, and industries. It is estimated that 16 million people used 802.11 in 2001, and this number will grow to 60 million by 2006. There are a variety of applications in wireless networks, such as military, environmental, medical and industrial applications. To develop these applications we normally use specific solutions for each case. This is a problem, making maintenance and extension a difficult task. Therefore, a tool that automatizes the effort of the programmer is of great utility.

This work presents a tool based on components, called BeanWatcher. This tool allows the development of monitoring applications for wireless networks. The applications developed execute in a mobile device that acts as monitoring element of remote data and/or as an actuator. To conceive this tool, a component model was used that standardizes the monitored elements. This model makes the applications simpler and portable by generating them in different environments (Java, J2me and C++).

As a case study, we developed applications for sensor networks. Such networks have the capacity of sensing and distributed processing. Factors related with the characteristics of the network, types and configurations of the sensors influence the development of the applications. Through BeanWatcher we developed applications for three different environments: medical monitoring, allowing the vital data of the patients to be seen in the doctor's PDA; industrial control, where the laborer can monitor the temperature of the equipment and control it remotely; and farm monitoring that allows the agriculturist to see data about the plantation. These applications have been developed to show the viability and efficiency of BeanWatcher.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Redes sem fio são um tipo especial de redes de computadores que utilizam ondas de rádio para prover comunicação entre os nós. Para estas redes existem alguns padrões tais como WLANs – *Wireless Local Area Networks* (IEEE 802.11), WPANs – *Wireless Personal Area Networks* (IEEE 802.15 and Bluetooth) e *Broadband Wireless Access* (IEEE 802.16) [15]. Estes padrões permitem que usuários sem fio utilizem sua rede em diferentes cenários como escolas e indústria. Estima-se que em 2001, 16 milhões de pessoas usaram o padrão IEEE 802.11 e que em 2006, este número irá crescer para 60 milhões [28].

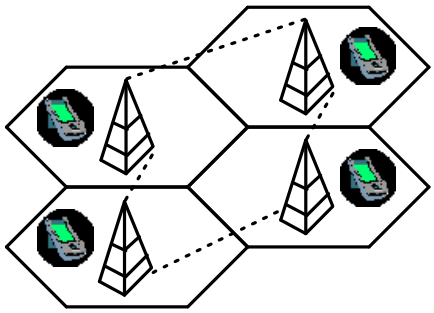
Dentro do contexto de redes sem fio encontramos as chamadas redes de sensores que é um grupo especial de redes *ad-hoc* [19] com capacidade de sensoriamento e processamento distribuído, podendo ser usada em uma grande variedade de aplicações, como por exemplo aplicações médicas, industriais, militares, meio ambiente e agropecuária [10, 30, 11].

Para melhor contextualizar as redes de sensores, em relação as outras redes de computadores, faremos um breve comentário de como funciona cada uma das diferentes redes sem fio existentes:

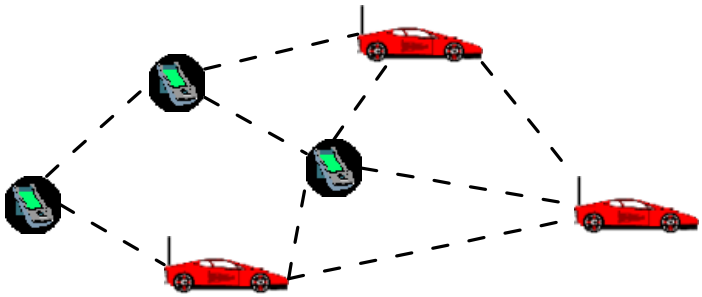
- **Redes estruturadas**: Possuem nós subordinados a uma estação base onde a comunicação entre dois elementos é feita através desta estação. Uma ilustração deste tipo de rede pode ser vista na Figura 1.1a.
- **Redes ad-hoc**: Não utilizam uma estação base para prover a comunicação entre dois nós. Neste caso a comunicação é feita utilizando os nós que estão entre a origem e o destino. Uma ilustração deste tipo de rede pode ser vista na Figura 1.1b.
- **Redes de sensores**: São redes que possuem a forma de comunicação como as redes *ad-hoc*, como pode ser visto na Figura 1.1c. Porém, além de outras particularidades, a comunicação destas redes é centralizada nos dados e os nós podem não possuir identificadores únicos.

1.1 Aplicações em Redes Sem Fio

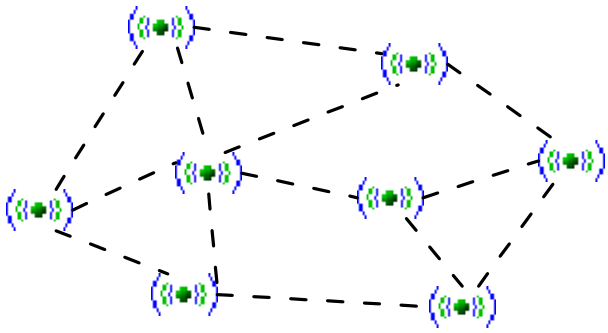
Como o presente trabalho trata da monitoração remota em ambientes sem fio, tentaremos ilustrar, com exemplos, algumas aplicações para estes ambientes, listadas a seguir:



(a) Rede estruturada.



(b) Rede ad-hoc.



(c) Rede de sensores.

Figura 1.1: Tipos de redes sem fio.

- **Aplicação médica**: Nesta aplicação um paciente é equipado com uma rede de sensores em seu corpo para coletar informações sobre batimentos cardíacos, nível de oxigenação do sangue, temperatura corporal e pressão arterial. O paciente e o médico teriam um celular que receberiam as informações desta rede de tal forma que o paciente poderia acionar um hospital caso estivesse sentindo-se mal e o médico, com as informações recebidas, poderia diagnosticá-lo a distância.

- **Agricultura de precisão**: Uma plantação pode ser equipada com diversos sensores coletando diferentes tipos de dados do ambiente, como temperatura, umidade e acidez do solo. Assim, um agricultor caminhando pela plantação pode receber os dados da rede em um dispositivo portátil com interface sem fio, visualizando toda a rede. A título de exemplo, o departamento de agricultura do Canadá em conjunto com o grupo de pesquisa da Intel, usando uma rede de sen-

sores de *motes*[1] [14] está sendo usada para monitorar a temperatura do ar em 50 hectares de vinhedo [1]. A partir de informações coletadas através da rede, o agricultor pode utilizar algumas técnicas para melhorar o crescimento das uvas e conseqüentemente a qualidade do vinho produzido.

- **Monitoração ambiental**: O centro de uma cidade equipado com uma rede de sensores, coletando informações sobre a qualidade do ar. Os cidadãos através de seus telefones celulares receberiam as informações coletadas, de tal forma que eles saberiam se aquela região, através dos poluentes existentes, estaria ou não prejudicando sua saúde.

- **Monitoração industrial**: Uma fábrica equipada com uma rede sem fio coletando diferentes tipos de dados dos equipamentos, como temperatura ou consumo de energia dos equipamentos. Assim, um funcionário, utilizando um dispositivo portátil, caminhando pela planta da fábrica receberia os dados dos equipamentos através de uma rede sem fio e poderia atuar nas máquinas, como por exemplo solicitar o desligamento, caso a temperatura ultrapasse um determinado valor.

Estas aplicações descritas acima, por executarem em dispositivos portáteis podem ser implementadas em diversas linguagens como J2ME [25] (distribuição de Java para dispositivos portáteis) ou SuperWaba [13] (linguagem destinada à aplicações em Palms). Uma abstração destas aplicações pode ser vista na Figura 1.2, onde teríamos um telefone celular monitorando uma determinada região ou equipamento.

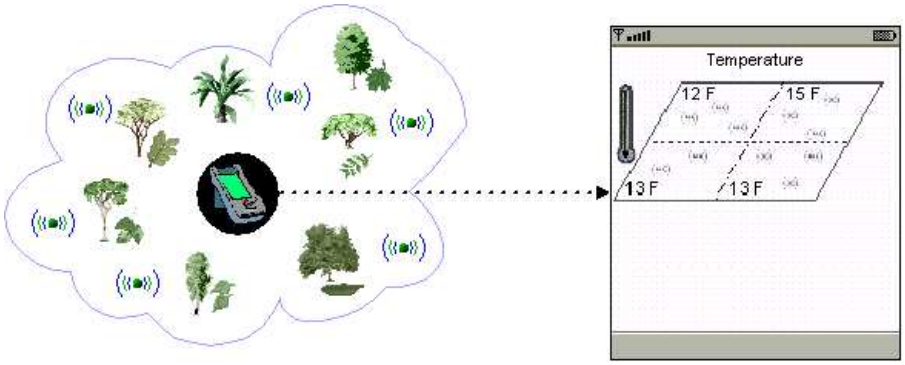


Figura 1.2: Exemplo de aplicação de monitoração.

Como podemos ver, as aplicações de monitoração podem ser tanto para ambientes sem fio de uma forma geral, como para redes de sensores. Seguindo a abordagem de ambientes sem fio, foi desenvolvido um protótipo do BeanWatcher com suporte a estes ambientes [22]. Em seguida outras funcionalidades foram inseridas ao BeanWatcher para permitir a monitoração em redes de sensores [21]. O presente trabalho utiliza, como estudo de caso, a geração de aplicações para monitoração em redes de sensores, porém pode ser utilizado para gerar aplicações em ambientes sem fio no geral.

[1]Nó comercial que tem o papel de sensor. Sua estrutura é composta por alguns sensores, um microprocessador, um rádio para prover a comunicação e uma bateria

1.2 Caracterização do Problema para Redes de Sensores

As aplicações de monitoração pode ser considerada em uma ou duas dimensões. As aplicações em uma dimensão são as que monitoram um único equipamento, ou seja, recebe apenas um *stream* de dados, por exemplo, monitoração de batimentos cardíacos ou da temperatura de um equipamento.

Já as aplicações em duas dimensões podem ser representadas através da monitoração de uma região, onde o valor recebido refere-se a combinação de um conjunto de *streams* de dados, que necessitem de um tratamento e um pré-processamento. Como exemplo de monitoração em duas dimensões temos as aplicações em redes de sensores.

Devido sua maior complexidade, optamos por utilizar, como estudo de caso, o desenvolvimento de aplicações de monitoração em duas dimensões no contexto de redes de sensores. Para melhor caracterizar os problemas encontrados no desenvolvimento destas aplicações, temos que considerar os seguintes aspectos:

- **Forma de comunicação**: Precisamos saber como os nós monitorados e o nó de monitoração trocam mensagens, ou seja, qual a tecnologia que é utilizada e qual o formato das mensagens.
- **Filtragem dos dados**: Como podemos ter diferentes elementos sendo monitorados em uma única aplicação é importante termos um mecanismo de filtragem dos dados para que a aplicação possa identificar qual tipo de dado está sendo monitorado.
- **Representação dos dados**: Como utilizamos diferentes mídias precisamos ter disponível diferentes formas de representar os dados recebidos da rede.
- **Mecanismos para permitir o funcionamento da rede**: No caso das redes de sensores o elemento de monitoração pode fazer parte da rede, logo é importante termos disponível mecanismos que permitem a interação entre o elemento de monitoração e os sensores.
- **Portabilidade as aplicações geradas**: Pela diversidade dos dispositivos que podem agir como elemento de monitoração como Palm, PC ou telefone celular é importante que as aplicações geradas sejam portáteis.

1.3 Motivação e Objetivos

Como foi observado na seção 1.1, as aplicações de monitoração são desenvolvidas com o objetivo de resolver problemas específicos, sem se preocupar com padronizações ou reaproveitamento de código. Isto causa problemas se tivermos interessados em dar manutenção ou estender estas aplicações. Além disso, se considerarmos a monitoração em duas dimensões outros fatores devem ser levantados, como foi observado na seção 1.2. Sendo assim, é de grande utilidade uma ferramenta que automatize a implementação destas aplicações permitindo um rápido desenvolvimento.

Partindo dessas dificuldades e do fato de não existir nenhum ambiente para desenvolvimento destas aplicações, apresentamos neste trabalho uma ferramenta baseada em componentes, chamada BeanWatcher, que utiliza um modelo de componentes para

padronizar os elementos pertencentes as aplicações em ambientes sem fio e possui suporte à geração de código em diferentes linguagens, oferecendo assim portabilidade às aplicações.

Com isso, o principal objetivo do trabalho é prover uma ferramenta com as seguintes características:

- **Portabilidade das aplicações geradas**: As aplicações geradas podem ser executadas em diferentes ambientes.
- **Reutilização de componentes**: Os componentes devem ser desenvolvidos de tal forma que possa ser reutilizados em outras aplicações.
- **Utilização de padrões e modelos**: Ao utilizar padrões e modelos o usuário terá uma maior flexibilidade no desenvolvimento e na manutenção das aplicações.
- **Fácil utilização**: Permitir que usuários com pouca experiência em programação desenvolvam aplicações complexas através da ferramenta.

Como contribuição do nosso trabalho podemos destacar o desenvolvimento de uma ferramenta gráfica, baseada em componentes, para a monitoração e atuação em ambientes sem fio, que pode ser utilizada por usuários com pouca experiência em programação. Além disso, as aplicações geradas podem rodar em dispositivos portáteis e o BeanWatcher pode ser estendido para dar suporte a geração de aplicações em diferentes linguagens.

1.4 Trabalhos relacionados

Do ponto de vista do nosso estudo de caso, redes de sensores, não foram encontradas ferramentas especificas para o desenvolvimento de aplicações de monitoração. Nas próximas subseções, temos alguns trabalhos que serviram de motivação para o desenvolvimento do BeanWatcher.

1.4.1 Pearl

Pearl, é uma ferramenta para criação de aplicações destinadas à monitoração de sistemas embutidos conectados em rede. A interface das aplicações geradas segue o paradigma de instrumentação virtual, onde os dados monitorados são apresentados por elementos gráficos semelhantes em aparência e funcionalidade aos instrumentos reais.

A base da ferramenta consiste em uma biblioteca de instrumentos virtuais responsáveis pela apresentação provenientes dos sistemas embutidos e possui protocolos de comunicação separando a camada de apresentação da camada comunicação. Estes protocolos manipulam os sistemas embutidos como elementos de redes comuns, permitindo a integração da instrumentação com a gerência da rede.

A construção das aplicações é realizada através de um editor gráfico que permite ao usuário criar, posicionar e configurar os instrumentos em um painel virtual. Na elaboração dos painéis, podem ser utilizados tanto os componentes disponíveis na biblioteca local quanto componentes disponíveis em sistemas embutidos conectados a Internet.

Esta ferramenta possui como principais vantagens a utilização de instrumentos remotos e a portabilidade das aplicações por gerar código Java. Porém, diferente do BeanWatcher, esta ferramenta não possui suporte a geração de aplicações de monitoração em redes de sensores e não suportam geração de código em outras linguagens, logo não podemos utilizar as aplicações geradas em dispositivos portáteis móveis.

1.4.2 LabVIEW

LabVIEW - *Laboratory Virtual Instrument Engineering Workbench* [16], desenvolvido pela *National Instruments* foi o primeiro sistema de instrumentação virtual comercial. A primeira versão do sistema, lançada em meados dos anos 80, utilizava a plataforma Macintosh. A escolha foi motivada pelo suporte gráfico propiciado pela plataforma, essencial para implementação dos instrumentos virtuais.

A arquitetura do LabVIEW é composta por três partes principais:

- **Módulo de acesso a dispositivos**: Consiste de um conjunto de *drivers* responsáveis pela leitura efetiva de equipamentos reais utilizando as interfaces de comunicação do computador, como interfaces seriais e de barramento. Os *drivers* disponibilizam os dados lidos através de uma interface de programação bem definida, fazendo com que as camadas responsáveis pelo tratamento e apresentação dos dados não necessitem conhecer detalhes de um dispositivo específico e possam tratar todas da mesma forma. O módulo de acesso aos dados não é programado pelo criador dos painéis, mas deve estar previamente instalado no sistema.

- **Diagrama de blocos do sistema**: Consiste no módulo onde é definido o fluxo de dados entre os instrumentos do painel virtual. Nesse diagrama, o criador dos painéis indica graficamente como os dados monitorados pelo sistema serão associados a instrumentos virtuais. Essa associação pode ser direta ou envolver funções lógica-aritméticas ou combinação de dados. A definição do fluxo de dados é realizada graficamente, conectando as entradas e saídas dos elementos por fios. A linguagem gráfica para a definição da lógica da aplicação é denominada linguagem G.

- **Painel frontal**: Consiste na interface gráfica propriamente dita. Todo instrumento definido no diagrama de blocos possui uma representação gráfica no painel frontal. Enquanto no diagrama de blocos apenas as entradas e saídas para os instrumentos são configuradas, no painel frontal apenas os atributos gráficos são configurados separando a lógica da interface da aplicação.

Esta ferramenta possui como principais vantagens a grande quantidade de instrumentos e *drivers* existentes para aquisição de dados e a fácil utilização para desenvolver aplicações. Porém, diferente do BeanWatcher, esta ferramenta não é destinada a geração de aplicações de monitoração em redes de sensores e apenas suporta geração de componentes externos em C, além dos componentes na linguagem nativa do LabVIEW.

1.5 Estrutura do Trabalho

O presente trabalho encontra-se dividido em seis capítulos, dos quais a presente introdução é o primeiro. O Capítulo 2 apresenta uma contextualização dos principais conceitos sobre redes de sensores e sobre o modelo de componentes utilizado.

O Capítulo 3 descreve o BeanWatcher, detalhando seus principais requisitos, como foi utilizado o modelo de componentes no contexto de aplicações para redes de sensores, qual a arquitetura e como ele é utilizado para desenvolver aplicações.

O Capítulo 4 discute o desenvolvimento de aplicações utilizando o BeanWatcher, focando no nosso estudo de caso. Além disso, mostraremos a estrutura dos componentes primitivos, dos componentes compostos e das aplicações geradas pela nossa ferramenta.

O Capítulo 5 mostra algumas aplicações desenvolvidas através do BeanWatcher.

Por fim, o Capítulo 6 apresenta algumas conclusões e as contribuições obtidas na realização do trabalho, além de futuras direções e possíveis extensões para o BeanWatcher.

Capítulo 2

Contextualização

Neste capítulo faremos uma contextualização sobre os principais conceitos utilizados no presente trabalho. Primeiramente falaremos sobre redes de sensores, uma vez que estas redes serão nosso estudo de caso. Em seguida discutiremos sobre as aplicações de monitoração para redes de sensores, onde levantaremos os principais requisitos destas aplicações. Por fim, mostraremos o modelo de componentes utilizado pelo BeanWatcher.

2.1 Redes de Sensores Sem Fio

Redes de sensores são um grupo especial de redes *ad-hoc* que possuem capacidade de sensoriamento e processamento distribuído, podendo ser usadas em uma grande variedade de aplicações, como monitoração ambiental, aplicações industriais, militares e agropecuárias.

Basicamente, estas redes são formadas por dispositivos compactos e autônomos, chamados de nós sensores. De acordo com [7] estes nós sensores são formados por quatro componentes básicos: Unidade perceptiva, unidade de processamento, transceptor e fonte de energia. Esta estrutura pode ser vista na Figura 2.1, onde os blocos tracejados: sistema de localização, mecanismo de mobilidade e gerador de energia, identificam complementos a estrutura dos sensores. Na unidade perceptiva temos um conversor de sinais analógicos para sinais digital (ADC) e um sensor. Na unidade de processamento, temos o processador e a memória.

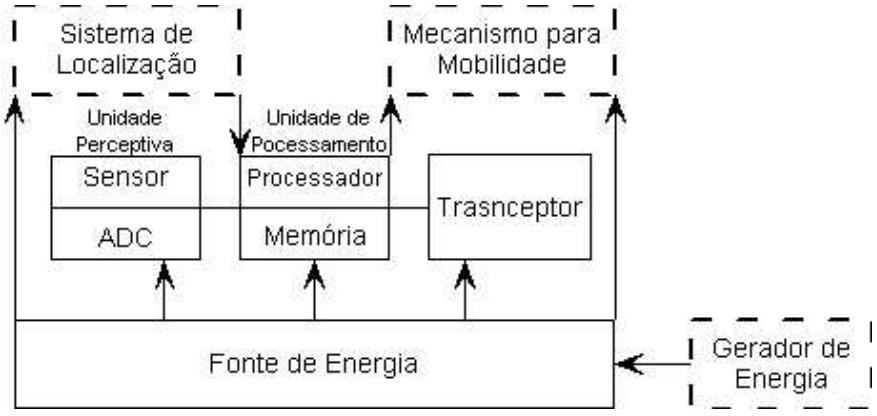


Figura 2.1: Estrutura de um nó sensor (extraído de [7]).

Além disso, os nós sensores coletam dados, processam localmente ou coordenadamente entre vizinhos podendo enviar a informação para o usuário. Por outro lado, as redes de sensores possuem outros elementos que são descritos a seguir:

- **Atuadores**: Possuem a função de atuar ou interferir no meio, a fim de corrigir falhas e/ou controlar o objeto monitorado. Fisicamente pode ser o próprio nó sensor.
- **Sorvedouros**: Também chamados de nós de monitoração, recebem os dados e os processam de forma a extrair alguma informação útil para o usuário. Além disso, os dados puros ou processados, podem ser enviados ao *gateway*.
- **Gateway**: Responsável por prover a comunicação da rede de sensores com outras redes de computadores. A mensagem percorre a rede de sensores até chegar ao *gateway* que encaminhará esta mensagem até o computador do operador, através de uma rede, como a Internet. Devemos destacar que em alguns casos os sorvedouros e *gateways* podem ser fisicamente o mesmo nó.

Uma vez conhecidos os elementos que compõem as redes de sensores, podemos falar sobre sua estrutura, onde basicamente temos os nós sensores distribuídos em uma área se comunicando e formando uma rede *ad-hoc*. Após isso, estes nós sensores interagem com o ambiente objetivando a coleta, o processamento e o envio dos dados a um ou mais nós sorvedouros. Em seguida, as informações coletadas pela rede são enviadas a um ou mais observadores através dos *gateways*. A comunicação entre os nós é realizada em múltiplos saltos. Esta estrutura de uma rede de sensores pode ser vista na Figura 2.2.

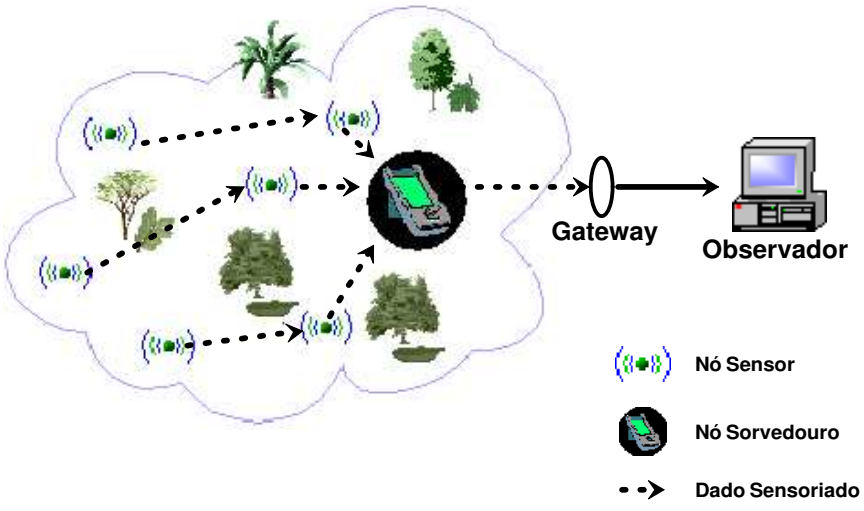


Figura 2.2: Estrutura de uma rede de sensores.

As aplicações das redes sensores possuem um caráter tanto de monitoração como de atuação. No caso das aplicações de monitoração, o sorvedouro apenas recebe os dados e os processam. Dentre os elementos que podem ser monitorados, podemos destacar temperatura, pressão, umidade e acidez. Nas aplicações de atuação, o sorvedouro pode enviar mensagens para os nós sensores interferirem no ambiente monitorado.

Para estas aplicações, temos que considerar as características das redes de sensores que são baseadas nos seus elementos, na topologia e na forma como os dados são coletados. Segundo [34], podemos caracterizar uma rede de sensores como:

- **Hierárquica** se a rede possui agrupamentos de nós, onde existe um líder que representa cada agrupamento ou **plana**, se a rede não possui agrupamento, ou seja, todos os nós se comunicam igualmente.
- **Homogênea** se os nós possuem a mesma configuração de *hardware* ou **heterogênea** se os nós possuem diferentes configurações de *hardware*.
- **Simétrica** se todos os nós possuem o mesmo raio de comunicação, por exemplo o nó 'A' alcança o nó 'B' e 'B' alcança 'A' ou **assimétrica** se os nós possuem diferentes raios de comunicação, por exemplo o nó 'A' alcança o nó 'B' mas 'B' não alcança 'A'.
- **Contínua** se os dados coletados são enviados continuamente ou **programada** se os dados são enviados obedecendo uma programação previamente estabelecida.
- **Dirigida a eventos** se a rede envia dados apenas quando ocorre algum evento ou **sob demanda** quando a rede permite a consulta parcial ou total aos nós.

2.2 Aplicações em Redes de Sensores

Apesar do potencial de suas aplicações, as redes de sensores possuem algumas restrições de recursos, tais como baixo poder computacional, largura de banda reduzida e especialmente fonte de energia limitada. Atualmente diversas linhas de pesquisas tem surgido para resolver estes problemas, como por exemplo a auto-organização da rede [33, 4], a disseminação do dado [20, 17] e o gerenciamento dos recursos da rede [31, 36, 35, 27].

As redes de sensores por serem um tipo específico de redes *ad-hoc* e utilizadas em ambientes hostis com condições imprevisíveis, devem ser auto-configuráveis, adaptáveis e possuir o gerenciamento escalável. Uma abordagem para as redes de sensores é o modelo centrado nos dados [20, 17], que permite a integração das operações da camada de aplicações com a camada de rede, oferecendo soluções mais eficientes.

Além disso, fatores relacionados com as características da rede, tipos e configurações dos sensores influenciam diretamente no desenvolvimento de tais aplicações. Porém, podemos diferenciar em alguns aspectos as aplicações que executam nos sensores das que executam nos sorvedouros. Em relação as aplicações que executam nos sensores, elas possuem uma forte dependência das características da rede, por outro lado as aplicações que executam nos sorvedouros podem ser projetadas de forma mais genérica por dependerem apenas de como os dados sensoriados devem ser tratados. Devemos destacar que como o BeanWatcher trata apenas da geração de aplicações de monitoração, do ponto de vista das redes de sensores as aplicações geradas na nossa ferramenta devem executar nos sorvedouros e no decorrer do texto quando falarmos sobre aplicações em redes de sensores estamos nos referindo à aplicações executando nos sorvedouros. Sendo assim, podemos classificar as aplicações que executam no sorvedouro em dois tipos:

- **Monitoração**: Os sorvedouros têm a função de disponibilizar os dados sensoriados para o usuário de forma gráfica ou textual. Neste tipo de aplicação, os dados são solicitados aos sensores periodicamente ou enviados sem solicitação caso ocorra um evento inesperado. Por exemplo, podemos considerar uma rede de sensores monitorando uma plantação de uva para coleta da temperatura do

ambiente, os dados são enviados para um dispositivo portátil pertencente ao agricultor e a temperatura é mostrada graficamente no dispositivo.

- **Atuação**: O usuário, além de fazer a monitoração a partir do sorvedouro, poderá enviar comandos para a rede interferindo no ambiente sensoriado. Por exemplo, a mesma rede para monitoração de uma plantação de uva, descrita no tópico anterior, sendo que o usuário solicita que a plantação seja irrigada após a identificação de altas temperaturas.

Após um levantamento das necessidades das aplicações que executam nos sorvedouros, observamos dois pontos importantes que devem ser tratados: a comunicação e o tratamento dos dados.

Estas aplicações devem suportar comunicação bi-direcional, concorrente ou seqüencial. Bi-direcional, pois o sorvedouro pode enviar mensagens continuamente para a rede solicitando dados ou pode interagir com a rede como atuador. Concorrente, o sorvedouro estará habilitado a receber e processar dados simultaneamente de vários sensores. Seqüencialmente, os dados são recebidos de forma seqüencial, o sorvedouro espera receber um dado de algum sensor, processa e só receberá outro dado após este processamento encerrar.

Caso as aplicações necessitem de algum tratamento nos dados, podemos utilizar agregação [36, 6, 20] ou mecanismos de fusão de dados [23, 3, 12]. Mecanismos de agregação são utilizados de forma simples e eficiente, pois os dados são apenas organizados para aplicação e nenhuma inferência é feita nestes dados. Já os mecanismos de fusão combinam diferentes dados para obter uma informação relevante ou para oferecer uma melhor qualidade aos dados e pode ser utilizado com cooperação, como proposto por Durrant-Whyte [9] onde diferentes nós contribuem com diferentes dados para executar uma tarefa em comum.

Como exemplo da utilização de fusão de dados consideremos a Figura 2.3 que mostra possíveis componentes em uma aplicação de rede de sensores, envolvendo fusão de dados. Nesta aplicação existem quatro níveis de processamento:

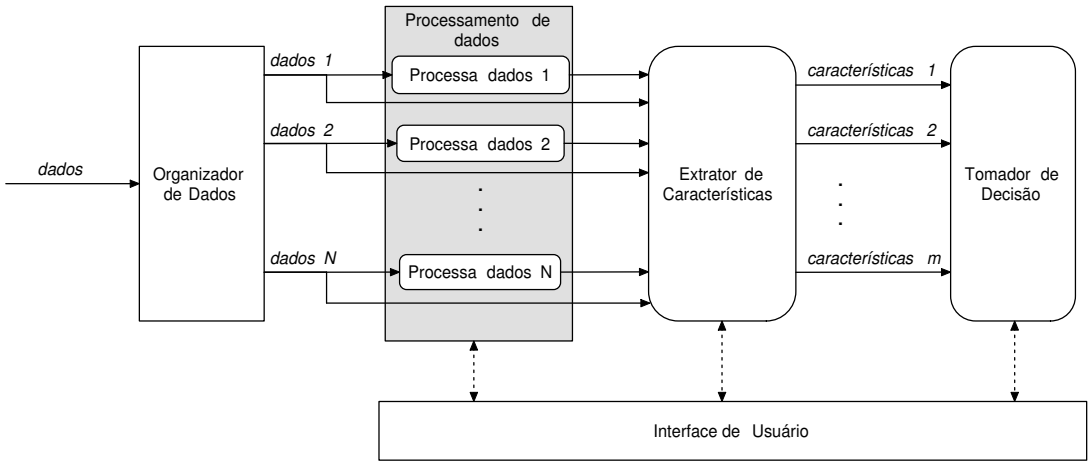


Figura 2.3: Estrutura de uma aplicação utilizando fusão de dados.

- **Organizador de Dados**: Responsável por receber vários dados através do mesmo canal de comunicação, então ele organiza e direciona estes dados para o módulo de processamento apropriado.

- **Processamento de Dados**: Responsável por executar a fusão de dados em baixo nível, por exemplo eliminar redundância ou filtrar os dados.
- **Extrator de Características**: Responsável por fundir os dados e obter características relevantes do ambiente, como por exemplo identificar o tipo de variável que está sendo monitorada e o que ela representa. Numa aplicação real teríamos a identificação de características como o solo está seco.
- **Tomador de Decisão**: Responsável por formular os planos de ação em resposta a uma situação identificada, por exemplo, tomar uma decisão sobre uma característica identificada. Numa aplicação real, poderíamos tomar a decisão de irrigar o solo ou apagar um incêndio.

2.3 Modelo de componentes

Antes de definirmos o modelo de componentes utilizado, devemos ter em mente os seguintes conceitos:

- **Interface de um componente**: Também chamada de protocolo do componente, é onde definimos como um componente conversará com outro. É a definição de como os dados são disponibilizados e recebido por um componente. Por exemplo, dois componentes 'A' e 'B' só podem interagir se a interface de saída de 'A' for compatível com a interface de entrada de 'B' ou vice-versa.
- **Comportamento de um componente**: É a definição de como o componente executará dentro da aplicação. Cada componente deve ter associado a ele um comportamento bem definido.

Devemos salientar que, para utilizarmos componentes nas aplicações, precisamos apenas saber qual o seu comportamento e interface.

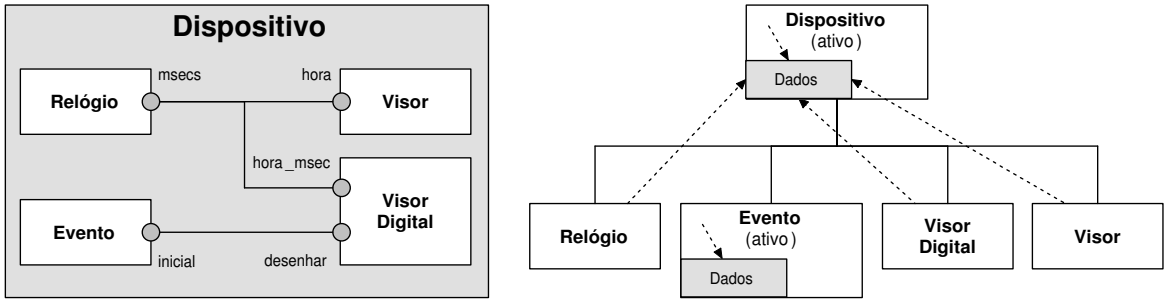
O modelo de componentes que utilizamos no BeanWatcher foi o PECOS - *PErvasive COmponent System* [29] que é um modelo destinado à aplicações em sistemas embutidos.

PECOS é dividido em dois sub-modelos: estrutural e de execução. O sub-modelo estrutural define as entidades incluídas no modelo, suas características e propriedades. O sub-modelo de execução define a semântica de execução dos componentes. Para ilustrar a utilização do PECOS consideremos a Figura 2.4, que representa o funcionamento de um relógio. Este exemplo será utilizado para explicar os sub-modelos do PECOS.

2.3.1 Sub-modelo Estrutural

Como foi citado o sub-modelo estrutural do PECOS define as entidades, características e propriedades do modelo. Existem três entidades principais neste sub-modelo:

Componentes : Cada componente tem uma semântica e comportamento bem definidos. Os componentes são usados para centralizar e organizar os dados e a computação da aplicação gerada. No exemplo mostrado na Figura 2.4a, os componentes são Dispositivo, Relógio, Visor, Evento e Visor Digital. Os componentes podem ser classificados em:



(a) Sub-modelo estrutural. (b) Sub-modelo de execução.

Figura 2.4: Aplicação modelada através do PECOS.

- **Passivos**: Estes componentes não controlam sua própria execução e são usados como parte do comportamento de outros. Na Figura 2.4a, os componentes passivos são Relógio, Visor e Visor Digital.
- **Ativos**: Estes componentes controlam sua própria execução e são usados para modelar atividades muito rápidas ou muito lentas, como leitura em registradores ou escrita em memória. Na Figura 2.4a, os componentes ativos são Dispositivo e Evento.
- **Eventos**: Estes componentes são similares aos ativos, porém sua execução é provocada por um evento. No nosso exemplo não temos componentes de eventos.
- **Simples**: Estes componentes são os componentes que possui um comportamento lógico associado a ele, em outras palavras as operações executadas nas aplicação ficam a cargo dos componentes simples. Componentes simples também são chamados de componentes primitivos.
- **Compostos**: Estes componentes são construídos usando sub-componentes, chamados de primitivos, conectados entre si, através de ligações que não são visíveis ao usuário. Além disso, um componente composto define um escalonamento específico e uma ordem exata de execução dos seus sub-componentes. Na Figura 2.4a, o componente composto é o Dispositivo.

Portas : Fornecem um mecanismo de interação entre os componentes, representa a interface do componente. Portas de saída são conectadas às portas de entrada, como ilustrado na Figura 2.4a, as portas de saída são msecs e inicial, e as de entrada são hora, hora_msec e desenhar. Além disso, as portas podem ser divididas em:

- **Portas de entrada**: Habilitam os componentes a receberem algum dado externo.
- **Portas de saída**: Permitem que os componentes disponibilizem dados a serem enviados para outros.
- **Portas de entrada/saída**: Habilitam o componente a receber e disponibilizar dados para outros.

Conectores : Descrevem o relacionamento de compartilhamento entre duas portas e são representados por linhas ligando dois componentes.

Além disso, ainda no sub-modelo estrutural, PECOS define as propriedades e os componentes pai. Propriedades são informações dos componentes, tais como uso de memória ou tempo de execução. A estrutura de uma aplicação gerada pelo modelo é sempre hierárquica onde o componente do topo é sempre um componente composto (pai).

2.3.2 Sub-Modelo de Execução

Como citado anteriormente, PECOS oferece um sub-modelo para a execução das aplicações, que descreve a semântica de execução dos componentes. Para isso é preciso definir como os dados são sincronizados entre os componentes executando em diferentes *threads*.

Pela sua forma de execução, as aplicações modeladas com o PECOS podem gerar problemas de sincronismo de dados. Um simples exemplo onde podemos identificar este problema seria se tivéssemos dois componentes ativos conectados através de uma porta, ambos podendo ler e escrever dados simultaneamente por diferentes operações. Para resolver este problema, PECOS habilita componentes ativos e de eventos a terem um espaço de dados privado, onde eles podem atualizar incondicionalmente e periodicamente um dado privado que é sincronizado pelo componente pai. Na Figura 2.4b é possível observar o espaço de dados privados nos componentes Dispositivo e Evento.

Devido a esta necessidade de sincronismo, componentes ativos e de eventos assumem dois possíveis comportamentos: execução e sincronismo. O de execução define as ações do componente quando este é executado. O de sincronismo especifica como o espaço de dados privados é sincronizado com o componente pai.

Uma vez definido como os dados são sincronizados entre os componentes, temos a semântica de execução obedecendo as seguintes regras:

- Execução de um componente passivo é feita por uma *thread* de seu componente pai.
- Sincronização é executada por uma *thread* de seu componente pai.
- Componentes ativo e de evento executam seus sub-componentes usando um controle de *thread*.
- Cada componente pai tem um escalonador para seu filho.

2.4 Considerações Finais

Neste capítulo abordamos os principais conceitos utilizados no BeanWatcher. Inicialmente falamos o que são as redes de sensores mostrando quais os principais elementos e características. Em seguida discutimos sobre os tipos de aplicações para redes de sensores, principalmente as que são executadas no sorvedouro, uma vez que nosso estudo de caso é direcionado a estes tipos aplicações. E por fim falamos do PECOS, que é o modelo de componentes utilizado pelas aplicações geradas através do BeanWatcher. No próximo capítulo falaremos, sobre a ferramenta BeanWatcher.

Capítulo 3

BeanWatcher

BeanWatcher é uma ferramenta gráfica baseada em componentes que tem como objetivo facilitar o desenvolvimento de aplicações para monitoração e atuação em ambientes sem fio. Estas aplicações possuem grandes desafios e vários requisitos que a nossa ferramenta tenta atender diminuindo o trabalho dos desenvolvedores.

Basicamente, o BeanWatcher possui as seguintes funcionalidades, que serão discutidas no decorrer do capítulo:

- Criação de novos componentes, não pertencentes ao repositório.
- Reimplementação do repositório existente para outras linguagens.
- Criação de componentes compostos através de uma área de trabalho.
- Criação de aplicações onde a linguagem alvo da aplicação pode ser escolhida pelo desenvolvedor.
- Edição de código de componentes já existentes ou implementação do comportamento de componentes criados.

Este capítulo segue com uma discussão sobre os principais requisitos da ferramenta. Em seguida, mostraremos como utilizamos o PECOS, para atender aos requisitos das aplicações de monitoração e atuação remota. Posteriormente, descreveremos a arquitetura e como foi desenvolvida a nossa ferramenta. Por fim, faremos uma exposição de como é a utilização do BeanWatcher.

3.1 Requisitos da Ferramenta

Além dos requisitos inerentes às aplicações de monitoração e atuação em ambientes sem fio, existem alguns requisitos relacionados ao BeanWatcher, ou seja, o que deve ser oferecido para que as aplicações nestes ambientes sejam criadas e gerenciadas. Temos os principais requisitos observados e como o BeanWatcher os atende, estão listados a seguir:

- **Forma de comunicação**: As aplicações em ambientes sem fio precisam ter um mecanismo de comunicação bem definido, de forma a permitir que os dados sejam solicitados periodicamente ou possam ser recebidos caso ocorra um evento inesperado na rede. Para isto, definimos no BeanWatcher uma entidade chamada

Comunicador que permite ao desenvolvedor utilizar a comunicação de forma simples e independente do elemento monitorado.

- **Filtragem dos dados**: Como podemos ter diferentes formato de mensagens para cada tipo de rede, definimos no BeanWatcher uma entidade responsável por tratar os dados recebidos pela rede, chamada Filtro, de forma que o desenvolvedor possa configurar esta entidade de acordo com as necessidades de sua aplicação.

- **Representação dos dados monitorados**: Como os dados monitorados não possuem uma padronização, cada desenvolvedor os implementa de uma forma, ocasionando pouca reutilização de código. Para evitar isto, utilizamos no BeanWatcher o modelo de componentes PECOS para padronizar as aplicações geradas.

- **Mecanismos para o funcionamento da rede**: As aplicações em ambientes sem fio, principalmente as redes de sensores possuem mecanismos, tais como: fusão, compressão e agregação de dados que devem ser suportados pelas aplicações. Para isto, o BeanWatcher permite criação de novos componentes para atuarem como estes mecanismos de forma que o usuário possa configurar qual algoritmo que ele deseja utilizar em sua aplicação.

- **Portabilidade**: Como as aplicações podem ser utilizadas em diferentes ambientes, podemos ter diferentes equipamentos e dispositivos atuando como elementos de monitoração, por exemplo um PC e um dispositivo portátil, através do BeanWatcher podemos gerar código da mesma aplicação em diferentes linguagens.

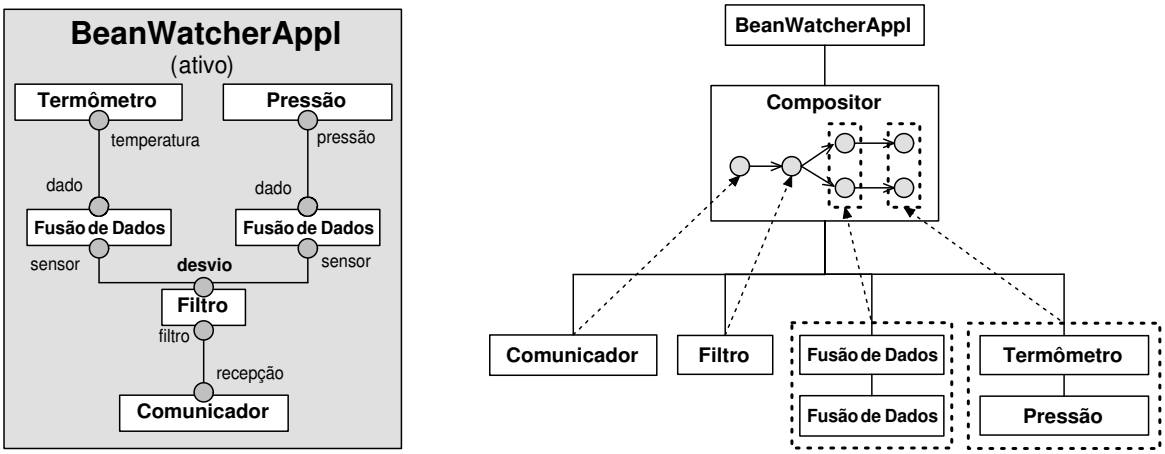
Os tópicos acima, descrevem os aspectos das aplicações de ambientes sem fio que tentamos cobrir com o BeanWatcher. Em resumo, devemos ter algum mecanismo de comunicação, um filtro ligado a este mecanismo para tratar os dados recebidos, um modelo que represente os elementos monitorados, prever a utilização de uma API com diversos mecanismos disponíveis e por fim oferecer portabilidade as aplicações geradas. E como pode ser visto o BeanWatcher cobre estas necessidades.

3.2 PECOS no BeanWatcher

Nesta seção, mostraremos como os elementos monitorados foram modelados a partir do PECOS e quais considerações foram feitas para que este modelo se adequasse as necessidades das aplicações de monitoração e atuação em ambientes sem fio.

Como vimos anteriormente o PECOS possui dois sub-modelos um estrutural e um de execução. Para um melhor entendimento de como utilizamos o modelo, consideremos o seguinte exemplo: uma aplicação que monitora a temperatura do solo e a pressão atmosférica, onde para cada um destes elementos monitorados mecanismos de fusão de dados são aplicados. Para modelar esta aplicação através do PECOS utilizamos os componentes Termômetro, Pressão, Fusão de Dados, Filtro e Comunicador. A ilustração deste exemplo, pode ser visto na Figura 3.1. Como podemos observar na figura utilizamos os sub-modelos referentes ao PECOS com as seguintes considerações:

Sub-modelo estrutural : Dois novos elementos foram inseridos, o Comunicador e o Filtro. O Comunicador foi criado para dar suporte a monitoração remota as



(a) Sub-modelo estrutural.

(b) Sub-modelo de execução.

Figura 3.1: Utilizando PECOS no BeanWatcher.

aplicações modeladas com o PECOS. O Filtro, é um componente responsável por receber os dados do Comunicador e tratá-los de tal forma que eles possam ser melhor visualizados pela aplicação. A ilustração do sub-modelo estrutural pode ser visto na Figura 3.1a.

Além disso, inserimos na estrutura da aplicação modelada pelo PECOS, uma entidade centralizadora, chamada Compositor, que é responsável por associar o componente ativo (pai) aos passivos (filhos). Todas as chamadas aos filhos e a resposta ao pai são feitas através dele. Esta entidade e a estrutura da aplicação pode ser vista na Figura 3.2. Onde temos os componentes Termômetro, Pressão e BeanWatcherAppl, associados ao Composer de forma que a comunicação entre os componentes filhos (Termômetro e Pressão) com o componente pai (BeanWatcherAppl), é feita através do Compositor.

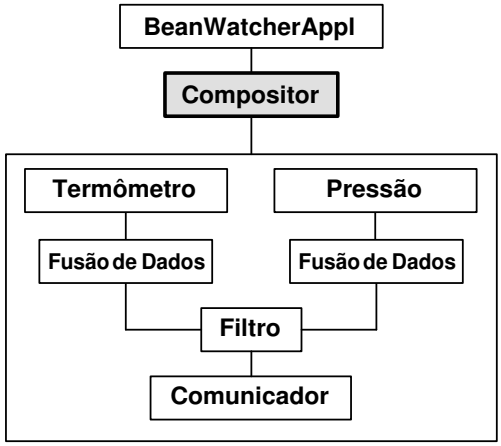


Figura 3.2: Sub-modelo estrutural com o Compositor.

Sub-modelo de execução : Não utilizamos o conceito de *multi-thread*, adotamos a execução em *pipeline*, ou seja, apenas um componente pode ser executado por vez. Isto é ilustrado na Figura 3.1b onde o componente BeanWatcherAppl

executa seqüencialmente os seus filhos. Porém, como o Compositor é o elemento centralizador, a execução dos filhos é feita através dele.

Como podemos observar foi necessário modificar e incrementar o PECOS inserindo os componentes Comunicador e Filtro e criando o elemento Compositor ao sub-modelo estrutural, além disso, adotamos a execução em *pipeline* no sub-modelo de execução. Estas considerações foram feitas para que pudéssemos utilizar o PECOS para modelar os elementos envolvidos nas aplicações de monitoração ou atuação em ambientes sem fio.

3.3 Arquitetura do BeanWatcher

Nesta seção discutiremos a arquitetura do BeanWatcher (Figura 3.3), onde mostraremos os principais módulos da ferramenta e como eles se comunicam.

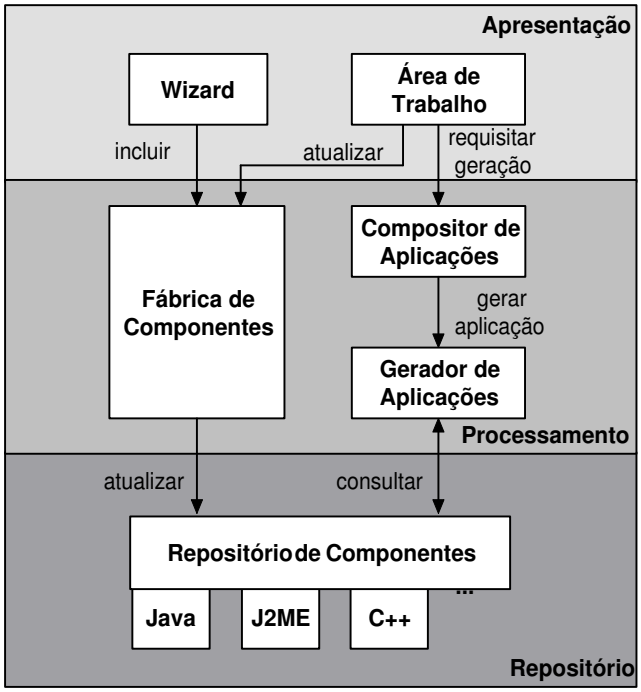


Figura 3.3: Arquitetura do BeanWatcher.

3.3.1 Módulo de Apresentação

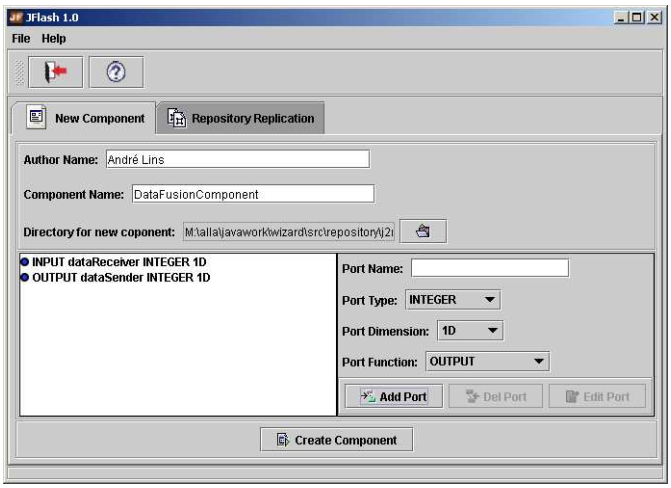
O módulo de apresentação, representa a interface do BeanWatcher com o desenvolvedor sendo composta por dois elementos: o *wizard* e a área de trabalho.

Wizard : É a parte do BeanWatcher responsável por criar novos componentes não pertencentes ao repositório. A criação pode ser realizada de forma individual, ou seja, apenas um componente é criado, ou todos os componentes podem ser reimplementados em outras linguagens, além das suportadas pelo repositório. Os componentes criados são sempre passivos e seguem o modelo definido pelo BeanWatcher. Outro aspecto do *wizard* é que apenas a estrutura dos componentes

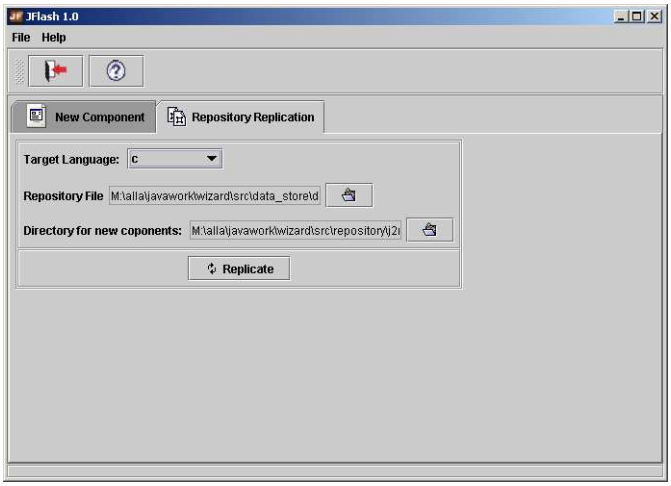
é gerada, sendo necessário que o comportamento seja implementado posteriormente.

O *Wizard* possui duas interfaces: uma para criação de novos componentes básicos e outra para reimplementar um repositório existente, conforme descrito a seguir:

- **Criação de novos componentes**: É responsável por coletar as informações passadas pelo usuário sobre o componente a ser criado e repassá-las para a Fábrica de Componentes. A interface para o módulo de coleta pode ser vista na Figura 3.4a.
- **Reimplementação**: Possibilita ao usuário a reimplementação automática do repositório. Como exemplo, podemos considerar a existência de um repositório em Java com 100 componentes e o desenvolvedor em um certo momento necessita de aplicações em C++. Para evitar a criação manual de cada componente basta solicitar a reimplementação para ter todo os componentes do repositório em C++. A interface para o módulo de reimplementação pode ser vista na Figura 3.4b.



(a) Criando um novo componente.



(b) Reimplementando o repositório.

Figura 3.4: Interface do *wizard*.

Área de Trabalho : É onde a aplicação é desenvolvida, através dela o usuário monta, configura e gera sua aplicação. Caso seja necessário, os componentes existentes podem ser alterados para atender melhor as necessidades do desenvolvedor.

Na Área de Trabalho existem duas interfaces: uma para a montagem de componentes e outra para edição de código, conforme descritas a seguir:

- **Montagem de componente**: Nesta área criamos as aplicações, selecionando a opção de montagem de uma aplicação e através do menu escolhendo os componentes que irão formar a aplicação, em seguida interligá-los através de conectores e configurar suas propriedades se necessário. Além disso, podemos montar componentes compostos que podem ser utilizados na aplicação, para isso selecionamos a opção de montagem de um componente e seguimos os mesmos passos para a montagem de uma aplicação. A área de montagem pode ser vista na Figura 3.5a.
- **Edição de código**: Nesta área implementamos o comportamento dos componentes criados através do *wizard* ou otimizamos algum componente existente, para isso basta selecionar o componente a ser editado que código o fonte será disponibilizado para modificações. Esta área pode ser vista na Figura 3.5b.

3.3.2 Módulo de Processamento

O módulo processamento é responsável por criar e atualizar os componentes ou montar e gerar as aplicações feitas através do BeanWatcher. Para isso temos três sub-módulos Fábrica de Componentes, Compositor de Aplicações e Gerador de Aplicações.

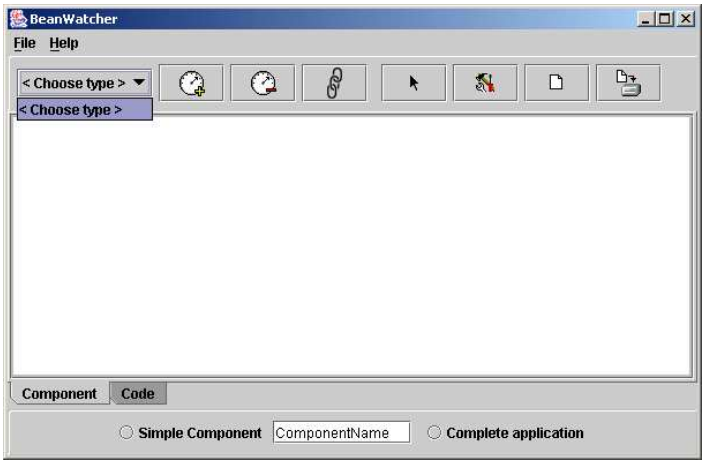
Fábrica de Componentes : É o sub-módulo responsável por criar novos componentes a partir dos *templates* suportados pela ferramenta e armazená-los no repositório do BeanWatcher. As informações utilizadas para a geração dos componentes são guardadas num arquivo de propriedades, para que no momento da reimplementação ou na geração das aplicações, estas informações sejam utilizadas. Além disso, este sub-módulo é responsável por atualizar os componentes que tiveram seu comportamento modificado na área de trabalho.

Compositor de Aplicações : É o sub-módulo responsável por coletar informações da área de trabalho, montar as dependências, propriedades e características dos componentes e repassar estas informações para o Gerador de Aplicações.

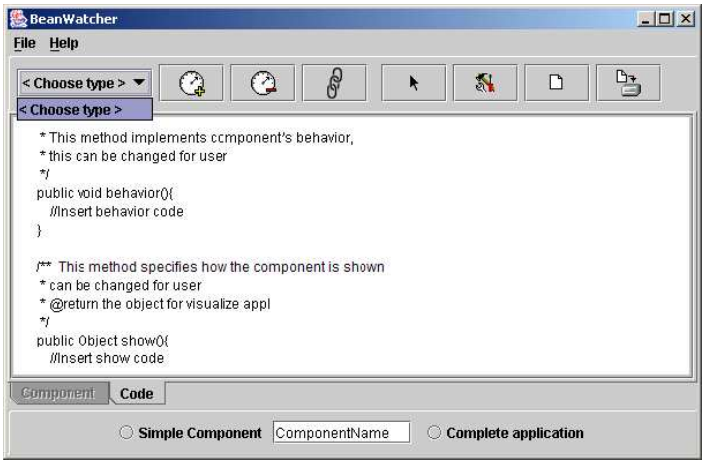
Gerador de Aplicações : Ao receber as informações de forma abstrata do Compositor de Aplicações, este sub-modulo gera a aplicação de fato. Para isso, os componentes existentes no repositório são copiados para o diretório da aplicação e em seguida o código referente aos componentes Compositor e Filtro é gerado. A partir deste ponto a aplicação está pronta para ser compilada e executada.

3.3.3 Módulo Repositório

O módulo repositório contém os componentes implementados nas linguagens suportadas. Como vimos anteriormente, este repositório é alimentado e atualizado através da Fábrica de Componentes e é utilizado pelo Gerador de Aplicações.



(a) Área de montagem.



(b) Área de edição.

Figura 3.5: Interface da área de trabalho.

O repositório é composto por *templates* de configuração para cada linguagem alvo suportada que são utilizados pelo *wizard* na criação de novos componentes e pelo BeanWatcher para a geração de novas aplicações.

Além disso, podemos reimplementar todos os componentes já existentes, pois como falamos anteriormente, as informações passadas pelo usuário na criação de um novo componente são armazenadas num arquivo de propriedades de forma a não precisarmos criar cada componente novamente. Porém, o comportamento de cada componente deve ser implementado através da área de edição de código.

A atual versão do BeanWatcher possui um repositório implementado em J2ME. Onde temos a seguinte classificação para os componentes:

- **Componentes de monitoração**: São os componentes utilizados para disponibilizar os dados coletados dos dispositivos. Na Tabela 3.1 temos a lista destes componentes com sua descrição e interface no BeanWatcher.

- **Componentes de atuação**: São os componentes utilizados para atuar nos dispositivos. Na Tabela 3.2 temos a lista destes componentes com sua descrição e interface no BeanWatcher.

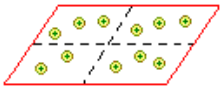
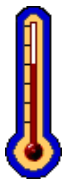
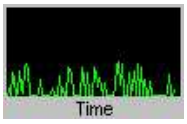
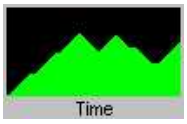
Ilustração	Nome	Descrição
	SensorNet	Componente que representa graficamente uma rede de sensores. Utilizamos a mesma interface para monitorar qualquer dado da rede, onde diferenciamos cada um dos elementos sensoriados através do ícone colocado na criação da aplicação. Além disso, todos os mecanismos utilizados na rede devem ser implementados como componentes funcionais.
	Thermometer	Componente para medir temperatura.
	DigitalNumber	Componente que representa números digitais. Pode ser utilizado para compor painéis digitais.
	Graph	Componente que representa um gráfico em duas dimensões.
	BarGraph	Componente que representa um gráfico em barras.

Tabela 3.1: Componentes de monitoração.

- **Componentes de alarme**: São os componentes utilizados para disparar algum alarme de acordo com algum evento programado. Na Tabela 3.3 temos a lista destes componentes com sua descrição e interface no BeanWatcher.
- **Componentes funcionais**: São os componentes utilizados para executar alguma funcionalidade nas aplicações e não são visualizados pelos usuário. Na Tabela 3.4 temos a lista destes componentes com sua descrição e interface no BeanWatcher.
- **Componentes de apoio**: São os componentes utilizados para montar a interface das aplicações como textos e símbolos. Na Tabela 3.5 temos a lista destes componentes com sua descrição e interface no BeanWatcher.

3.4 Utilizando o BeanWatcher

Esta seção tem como objetivo mostrar a utilização do BeanWatcher. Para isso, criaremos um componente através do *wizard*. Em seguida, descreveremos como um

Ilustração	Nome	Descrição
	SquareButton	Componentes utilizados para atuação. Possuem a mesma funcionalidade modificando apenas na interface para representar o botão.
	RoundButton	
	Switcher	
	KeyButton	

Tabela 3.2: Componentes de atuação.

Ilustração	Nome	Descrição
	LedStop	Componente que representa um *led* vermelha, pode ser utilizado para indicar alarme.
	LedWait	Componente que representa um *led* laranja, pode ser utilizado para indicar advertência.
	LedGo	Componente que representa um *led* verde, pode ser utilizado para indicar área sem problemas.

Tabela 3.3: Componentes de alarme.

repositório pode ser reimplementado para uma nova linguagem alvo inserida. Por fim, mostraremos como as aplicação são montadas através do BeanWatcher.

Para ilustrar a utilização do BeanWatcher, consideremos o exemplo para monitoração da temperatura de uma plantação de uva onde a rede realiza a fusão dos dados antes de enviá-los para o sorvedouro. Além disso, temos um *Led* indicando se a temperatura passou de um determinado valor e um botão para indicar que a plantação deve ser irrigada. Como não temos um mecanismo de fusão no BeanWatcher, criaremos este componente através do *wizard*. Além disso, teremos um PC e um dispositivo portátil como sorvedouro, logo precisaremos da mesma aplicação em J2ME e C++, para isso o repositório deverá ser reimplementado.

3.4.1 Criando um Novo Componente

Para criar o componente *DataFusion* através da interface mostrada na Figura 3.4a, devemos passar as seguintes informações:

- **Nome do componente**: Para podermos recuperar este componente no repositório posteriormente. No nosso exemplo será *DataFusion*.
- **Caminho para o arquivo de propriedades**: Para que as informações possam ser armazenadas nele. No nosso exemplo é c:/temp.
- **Especificação das portas**: Informamos o número, direção (entrada/saída), nome, tipo e dimensão das portas do componente. No nosso exemplo teríamos

Ilustração	Nome	Descrição
	Communicator	Componente responsável por tratar a comunicação.
	Filter	Componente responsável por tratar os dados da aplicação.
	DataFusion	Componente responsável por executar fusão de dados.
	Alarm	Componente responsável por fazer o controle de alarme.

Tabela 3.4: Componentes funcionais

Ilustração	Nome	Descrição
	DigitalToken	Componente que representa separadores em *displays* digitais. Por exemplo, barras, vírgula, ponto, dois pontos etc.
	Label	Componente que permite a inserção de texto na aplicação.

Tabela 3.5: Componentes de apoio

duas portas, uma de entrada, chamada *inPort* e outra de saída, chamada *outPort*.

Com isso temos o componente *DataFusion* criado, porém devemos salientar que o seu comportamento deve ser implementado.

3.4.2 Reimplementando o Repositório

No nosso exemplo, precisaremos das aplicações geradas em C++. Como o BeanWatcher não possui suporte a tal linguagem, devemos criar e inserir os *templates* necessários para C++. A reimplementação é feita através da interface da Figura 3.4b, e segue os seguintes passos:

- Informar a nova linguagem a ser inserida, no nosso exemplo C++.
- Especificar a localização do arquivo de propriedades, para que ele seja utilizado para reimplementar, de forma automática, os componentes já existentes.
- Indicar o local onde os componentes serão armazenados.

Com isso, podemos ter aplicações geradas tanto em C++ como em Java. Devemos destacar que qualquer componente inserido na ferramenta poderá ser utilizado tanto para aplicações em C++ quanto Java, desde que seu comportamento esteja implementado.

3.4.3 Edição de código

Uma vez que criamos o componente *DataFusion* e reimplementamos o repositório para C++, devemos editar o comportamento destes componentes através da área de trabalho utilizamos a edição de código. Esta área pode ser vista na Figura 3.5b.

3.4.4 Montando uma aplicação

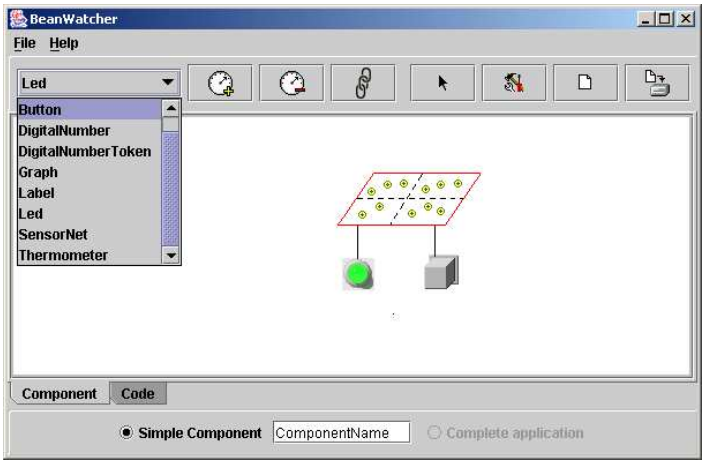
Após criado o componente *DataFusion*, reimplementado o repositório para C++ e editado o comportamento dos novos componentes, devemos montar a aplicação. Seguindo os seguintes passos:

- Selecionar a opção de montagem de componente e informar o nome do componente a ser gerado. No nosso exemplo montaríamos o componente *Temperature*. Esta opção pode ser vista na Figura 3.6a.
- Escolher os componentes básicos através do menu, inseri-los na área de trabalho e interligá-los através de conectores. No nosso exemplo escolheríamos os componentes *Temperature*, *Led* e *Button*. Esta escolha pode ser vista na Figura 3.7a. Após isso, o componente *Temperature* poderá ser utilizado na geração de aplicações.
- Selecionar a opção de montagem de aplicação. A partir daí, a aplicação será gerada. Esta opção pode ser vista na Figura 3.6b.
- Escolher os componentes através do menu, inseri-los na área de trabalho e interligá-los através conectores. No nosso exemplo escolheríamos os componentes *Temperature* e *DataFusion* (Figura 3.7b). Como fazemos monitoramento remoto, precisaremos dos componente *Filter* e *Communicator* na aplicação.
- Salvar a aplicação. Após isso, podemos compilar e executar a nossa aplicação.

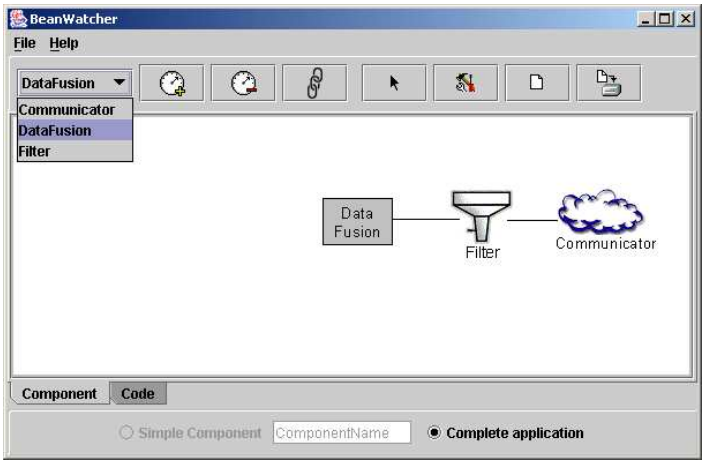
(a) Opção para montagem de componente.

(b) Opção para montagem de aplicação.

Figura 3.6: Opções da área de trabalho.



(a) Montagem de componente.



(b) Montagem de aplicação.

Figura 3.7: Montagem na área de trabalho.

3.5 Considerações Finais

Neste capítulo vimos quais os requisitos necessários para o desenvolvimento do BeanWatcher. Em seguida, mostramos como o PECOS foi utilizado para modelar os elementos sensoriados num ambiente sem fio e detalhamos a arquitetura da nossa ferramenta. Por fim, descrevemos como é a utilização do BeanWatcher no desenvolvimento de uma aplicação.

Capítulo 4

Desenvolvimento de Aplicações Utilizando o BeanWatcher

Neste capítulo falaremos sobre os requisitos e desenvolvimento de aplicações de monitoração e atuação através do BeanWatcher. Em seguida, discutiremos como uma aplicação é desenvolvida. Mostraremos como é a estrutura das aplicações geradas através do BeanWatcher. Por fim, descreveremos a utilização do BeanWatcher em um processo de desenvolvimento de software. Como utilizamos redes de sensores como estudo de caso, o presente capítulo será direcionado para estas redes.

4.1 Requisitos das Aplicações de Monitoração e Atuação em Redes de Sensores

Tradicionalmente, desenvolvimento de sistemas são organizados em arquitetura com diferentes camadas lógicas, onde o sistema é dividido em níveis de abstração que partem desde a camada física até a camada de aplicação, este modelo é usado de forma *bottom-up*. Por outro lado, redes de sensores são direcionadas às aplicações, sendo mais adequada uma abordagem *top-down*, como sugerido por [31], pois uma vez que os requisitos de negócios são conhecidos os de níveis mais baixos se tornam mais claros.

Porém, para desenvolver aplicações para redes de sensores os projetistas devem pensar nas seguintes questões:

- **Finalidade da rede**: Que tipo de dado está sendo sensoriado e qual o objetivo da rede? As aplicações deverão ser capazes de mostrar qualquer dado sensoriado, por exemplo, se a rede coletar temperatura, a aplicação deverá mostrar qual a temperatura no momento.
- **Visualização dos dados**: Qual a melhor forma de disponibilizar os dados ao usuário? Se a rede coletar informações de temperatura, a aplicação poderia mostrar um termômetro indicando a temperatura sensoriada.
- **Informações gerenciais**: Quando e quais informações gerenciais devem ser obtidas da rede? Por exemplo, a aplicação pode necessitar de informações sobre o mapa de energia ou localização dos nós.

- **Mecanismos de processamento distribuído**: Quais mecanismos estão sendo utilizados pela rede para pré-processar o resultado do sensoriamento? Por exemplo, poderíamos ter mecanismos de compressão, fusão ou agregação de dados, logo as aplicações devem suportar tais mecanismos.

- **Atuação**: Quais os parâmetros que devem ser passados para a rede no momento da atuação? Devemos nos preocupar em como devem ser passados os dados para a rede. Este fator está fortemente ligado ao propósito da rede.

As questões acima levantadas dizem respeito aos aspectos gerais das redes de sensores. Porém existem outros pontos que dependem exclusivamente do objetivo da rede, por este motivo apenas no momento de desenvolver as aplicações é que podemos cobrir estes aspectos, como:

- **Formato das mensagens da aplicação**: Como os dados serão enviados dos sensores? Qual o formato das mensagens enviadas? Como prever o sensoriamento de diferentes tipos de dados? Qual o formato das mensagens que os sensores recebem, no caso de termos atuação por parte dos sorvedouros?

- **Formato dos dados sensoriados**: Como os dados devem ser sensoriados e enviados? Por exemplo, se tivermos uma rede que coleta dados sobre a temperatura, podemos estar interessado em um intervalo ou em um simples valor da temperatura.

- **Mecanismo para melhoria do funcionamento da rede**: Como e quando são aplicados os mecanismo de melhoria do funcionamento da rede?

Claramente, as aplicações tendem a apresentar diferentes características e restrições. Desta forma, não é fácil oferecer uma única API para suportar as necessidades de todas aplicações. Uma boa abordagem seria considerar cada elemento desta aplicação como um componente que pode ser utilizado por diferentes aplicações, como feito pelo BeanWatcher.

4.2 Desenvolvendo Aplicações em Redes de Sensores

Nesta seção, mostraremos como uma aplicação em redes de sensores é desenvolvida sem a utilização de ferramentas para auxiliar e descreveremos como é o desenvolvimento da mesma aplicação através do BeanWatcher, apontando algumas vantagens da utilização desta ferramenta.

Baseado nos requisitos apresentados na seção anterior, podemos pensar em alguns cenários que mostram como estes requisitos são mapeados para o desenvolvimento de aplicações de monitoração.

Cenário 1 : Uma rede de sensores com a finalidade de visualizar a temperatura ambiente em uma plantação de uva.

- **Sem o BeanWatcher**: O desenvolvedor implementaria a visualização da temperatura e a comunicação.

- **Com o BeanWatcher**: Utilizaríamos o componente que representa uma rede de sensores para temperatura e o componente de comunicação, ambos disponíveis na ferramenta, e montaríamos a aplicação com estes componentes.

Cenário 2 : Agora a mesma aplicação, sendo que os dados precisam ser fundidos a cada 3 dados recebidos.

- **Sem o BeanWatcher**: O desenvolvedor reaproveitaria a aplicação do cenário 1. Porém, implementaria o suporte à fusão de dados a cada 3 dados recebidos, caso seja necessário modificar para 5 dados recebidos terá que reimplementar.
- **Com o BeanWatcher**: Utilizaríamos o componente que representa uma rede de sensores para temperatura e o componente para fusão de dados, em seguida configuraríamos a fusão a cada 3 dados recebidos, ou se necessário a cada 5.

Cenário 3 : Agora, além da temperatura ambiente, a rede passa a monitorar a umidade relativa do ar, soa um alarme quando a temperatura está acima de um determinado valor e mecanismos de compressão são aplicados.

- **Sem o BeanWatcher**: O desenvolvedor implementaria o comportamento da temperatura, da pressão, do alarme e o mecanismo de compressão.
- **Com o BeanWatcher**: Utilizaríamos os componentes: temperatura, pressão, alarme e compressão. Em seguida, configuraríamos as propriedades do alarme obedecendo as regras solicitadas pelo desenvolvedor e o mecanismo de compressão para suportar um determinado algoritmo.

Cenário 4 : Por fim, ao disparar o alarme, o agricultor solicita a irrigação da plantação para diminuir a temperatura.

- **Sem o BeanWatcher**: O desenvolvedor teria que integrar a aplicação feita no cenário 4 à solicitação de atuação nos sensores e para isso implementaria o mecanismo de atuação através de um botão.
- **Com o BeanWatcher**: Incluiríamos o componente para atuação que seria representado por um botão.

Além disso, para cada um dos cenários as aplicações poderiam ser executadas em um *desktop* e um dispositivo portátil. Neste caso, o desenvolvedor teria que reimplementar as aplicações para as duas plataformas, mas através do BeanWatcher, como a ferramenta suporta diferentes linguagens, bastaria gerar as aplicações para uma linguagem alvo diferente.

Note que, ao utilizar o BeanWatcher, o desenvolvedor não precisa conhecer como os componentes são implementados e nenhum código precisa ser implementado. Contudo, é permitido ao desenvolvedor alterar os componentes existentes com o objetivo de otimizá-los.

4.3 Estrutura das aplicações geradas através do BeanWatcher

Como vimos na seção anterior, o BeanWatcher possui grandes vantagens se for utilizado no desenvolvimento de aplicações de monitoração. Nesta seção mostraremos qual a estrutura interna dos componentes do repositório, dos componentes compostos e da aplicação final.

4.3.1 Estrutura dos componentes do repositório

Todos os componentes do repositório seguem o modelo PECOS adaptado para o BeanWatcher. Nesta seção mostraremos a estrutura dos componentes SensorNet, Comunicador e Filtro, apontando suas principais propriedades.

SensorNet : Este componente representa graficamente uma rede de sensores. Como pode ser visto na Figura 4.1, temos acima a representação abstrata do componente a direita a interface no BeanWatcher e abaixo a sua estrutura interna que possui apenas uma classe com as seguintes propriedades:

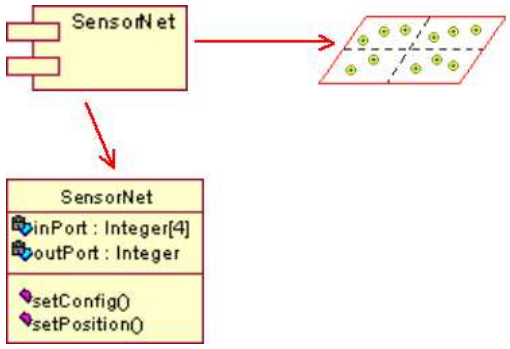


Figura 4.1: Estrutura do componente SensorNet.

- **Atributos**: Uma porta de entrada, *inPort*, que representa os valores de cada quadrante da rede de sensores. Uma porta de saída, *outProt*, que representa uma temperatura que pode ser repassada para outro componente. Por exemplo uma aplicação que precise soar um alarme caso a temperatura da rede ultrapasse um determinado valor.
- **Métodos**: Esta classe possui dois métodos *setConfig* e *setPossition*. Ambos são utilizados para atribuir os valores iniciais do componente que são passados pelo usuário no momento da montagem de componentes compostos, através da janela de propriedades. Uma ilustração da janela de propriedade deste componente pode ser vista na Figura 4.2, onde temos as seguintes propriedades:
  - **IconFile**: Indica o caminho para o ícone associado a este componente.
  - **Unit**: Indica a unidade utilizada para este componente, por exemplo, C ou F.
  - **NumberDataReceiver**: Indica o número de dados em relação aos quadrantes que será recebido pela aplicação.

– **IntervalValues**: Indica o intervalo de dados que será recebido, por exemplo, o intervalo de temperatura em um determinado tempo.

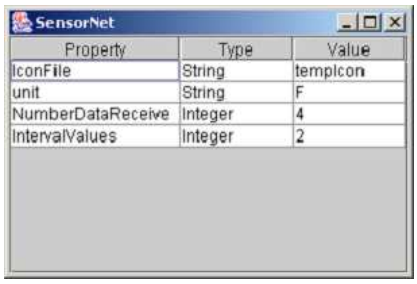


Figura 4.2: Janela de propriedades do componente SensorNet.

- **Comportamento**: Como este componente é visual o comportamento dele, implementado em J2me, através da chamada a uma método *paint*. A ilustração a nível de código para o comportamento pode ser visto na Figura 4.3.

```
/**
 * Behavior for this component
 */
public void behavior(){
    repaint();
}

/**
 * The default paint method
 */
public void paint( Graphics g ) {
    // Code insert
}
```

Figura 4.3: Implementação do comportamento do componente SensorNet.

Comunicador : Este componente é responsável por prover a comunicação das aplicações geradas. Como ilustrado na Figura 4.4 temos acima a representação abstrata do componente, a direita a interface no BeanWatcher e abaixo a sua estrutura interna que possui apenas uma classe com as seguintes propriedades:

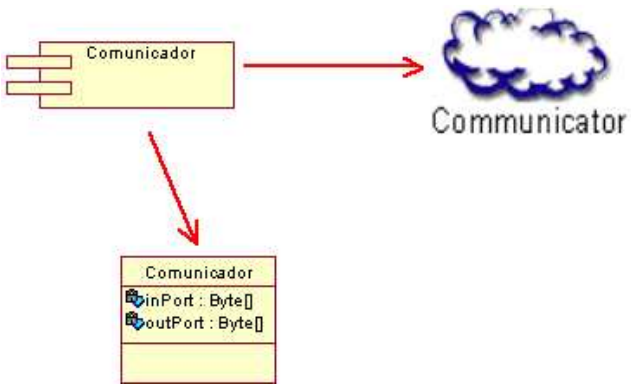


Figura 4.4: Estrutura do componente Comunicador.

- **Atributos**: Uma porta de entrada, *inPort*, que representa os valores recebidos dos elementos monitorados. Uma porta de saída, *outPort*, que representa os dados que devem ser repassados para um componente da aplicação.
- **Comportamento**: Como este componente é um componente funcional o comportamento dele é implementado, em J2me, normalmente sem a utilização de chamadas a outros métodos. A ilustração a nível de código para o comportamento pode ser visto na Figura 4.5

```
/**
 * Beahavior for this component
 */
public void behavior() {
    // Add code, use other classes
}
```

Figura 4.5: Implementação do comportamento do componente Comunicador.

Filtro : Este componente segue a mesma estrutura do componente de comunicação. Porém, devemos destacar que é ele quem trata os dados recebidos os direcionando para o devido componente da aplicação, logo carrega as características referentes a aplicação. Por este motivo o componente Filtro deve ser reimplementado sempre que um novo tipo de aplicação for criada. Para o nosso estudo de caso, utilizamos o filtro de tal forma que os dados recebidos da rede devem vir com um identificador da aplicação, onde o Filtro encontra em uma tabela estática qual a aplicação que deve tratar estes dados. Já no envio dos dados, a aplicação faz uma requisição para o Filtro que identifica o endereço de rede que este dado deve ser passado.

4.3.2 Estrutura dos componentes gerados

Através do BeanWatcher, para utilizarmos algum componente composto na aplicação final, devemos criá-lo a partir dos componentes simples do repositório. Nesta seção mostraremos a estrutura dos componentes compostos gerados.

Como ilustração, considere uma aplicação que monitore a temperatura de um equipamento que ao ultrapassar um determinado valor dispara um alarme e o equipamento possa ser desligado através de um botão.

Para desenvolver esta aplicação, precisamos montar um componente composto, onde o chamaremos de Componente Temperatura. Este componente montado no BeanWatcher, pode ser visto na Figura 4.6, onde os componentes simples Termômetro, Alarme, Led e Botão foram utilizados. Este componente ao receber dados da aplicação referente a temperatura irá mostrar este dado no termômetro e caso a temperatura ultrapasse o valor informado através das propriedades, minScale e maxScale, o alarme é acionado e o Led ficará piscando. O Botão será configurado com o endereço do equipamento monitorado para que quando o usuário o pressione, um mensagem seja enviada solicitando o desligamento.

Uma vez gerado o Componente Temperatura ele possui a estrutura apresentada na Figura 4.7, onde temos acima a representação abstrata do componente a direita a interface no BeanWatcher e abaixo as classes que o compõe. Este componente tem os seguintes elementos:

- **Classes do componente gerado**: Termômetro, Botão, Alarme e Led todas representando os componentes simples pertencentes ao repositório.

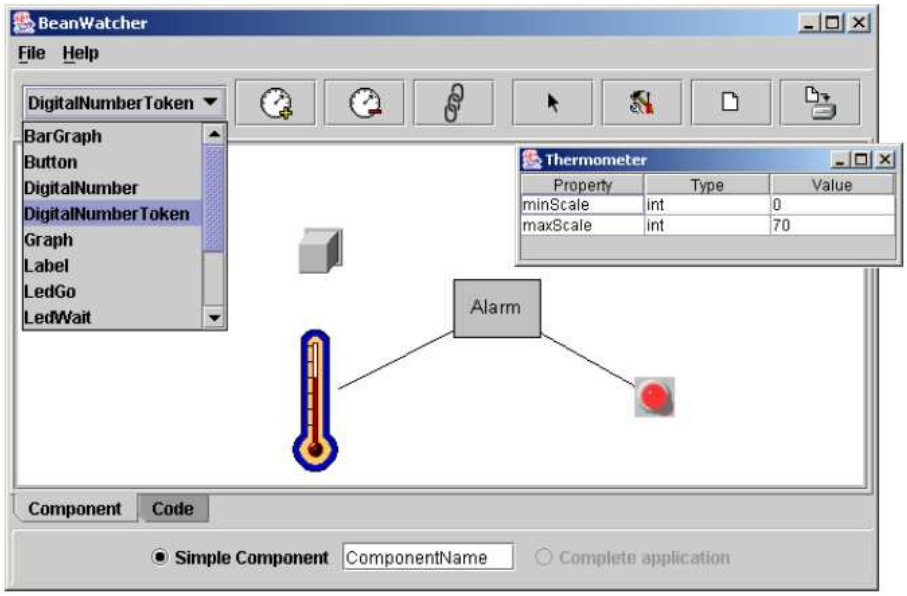


Figura 4.6: Componente Temperatura montado no BeanWatcher.

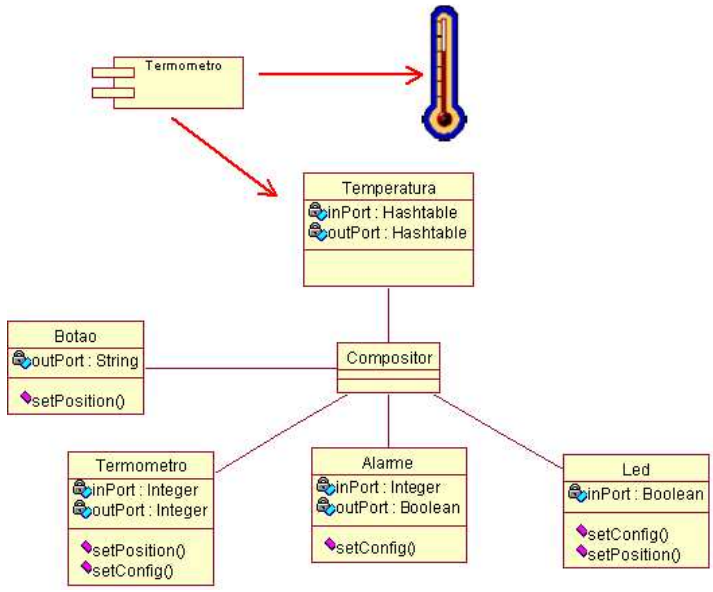


Figura 4.7: Estrutura do Componente Temperatura.

Classe Temperatura representa o componente pai do componente composto gerado. Possui como interface de entrada o atributo *inPort* e de saída *outPort*, esta classe sempre possui a mesma implementação para qualquer componente gerado pelo BeanWatcher, mudando apenas o nome.

Classe Compositor pertence a todos componentes compostos gerados através do BeanWatcher e sua geração é de forma automática. Sua função é centralizar a execução do componente gerado repassando as requisições do componente pai aos componentes filhos e as respostas dos filhos ao pai.

- **Configuração dos componentes através do Compositor**: A classe Compositor possui em seu construtor as informações de cada componente, obtidas

através da janela de propriedades. A implementação do construtor do Compositor para o exemplo dado pode ser visto na Figura 4.8, note que os componentes utilizados são inicializados e a os valores das propriedades são atribuídos para cada componente.

```
/** Creates a new instance of Composer */
public Composer() {
    this.objCanvas = new ObjectCanvas();

    Thermometer1 = new Thermometer();
    Alarm2 = new Alarm();
    LedStop3 = new LedStop();

    Thermometer1 .setConfig(0, 70);
    Alarm2 .setConfig(20, Alarm.POSITIVE);
    LedStop3 .setConfig();
}
```

Figura 4.8: Código gerado para o construtor do Compositor.

- **Comportamento do Compositor**: Na Figura 4.9 temos a ilustração do comportamento da classe Compositor gerado pelo BeanWatcher, é através de seu comportamento que o componente pai interage com os componentes filhos. Note que os componentes recebem informações a partir do método **setInPort()**, executam seu comportamento através do método **behavior()** e repassa a informação após seu processamento com o método **getOutPort()**.

```
/** Method for set behavior of composer */
public void behavior(){
    Thermometer1 .setInPort((Integer)inPort.get("Thermometer"));
    Thermometer1 .behavior();

    Alarm2 .setInPort(Thermometer1.getOutPort());
    Alarm2 .behavior();

    LedStop3 .setInPort(Alarm2.getOutPort());
    LedStop3 .behavior();

    Button4 .behavior();

    objCanvas.addElement(Thermometer1, 20, 15);
    objCanvas.addElement(LedStop3, 90, 15);
    objCanvas.addElement(Button4, 90, 50);
}
```

Figura 4.9: Código gerado para o comportamento do Compositor.

4.3.3 Estrutura da aplicação gerada

Na geração das aplicações utilizamos componentes compostos gerados anteriormente em conjunto com componentes simples. Para gerar a aplicação de temperatura sugerida na seção anterior deveríamos utilizar o Componente Temperatura e os componentes simples Filtro e Comunicador. Uma ilustração desta aplicação montada no BeanWatcher pode ser vista na Figura 4.10.

Cada um dos componentes utilizados na aplicação possui a estrutura mostrada em seções anteriores e a estrutura da aplicação gerada pode ser vista na Figura 4.11. O Compositor possui a mesma função do compositor para os componentes compostos e a estrutura da aplicação condiz com a sugerida pelo modelo PECOS utilizado(Figura 3.2).

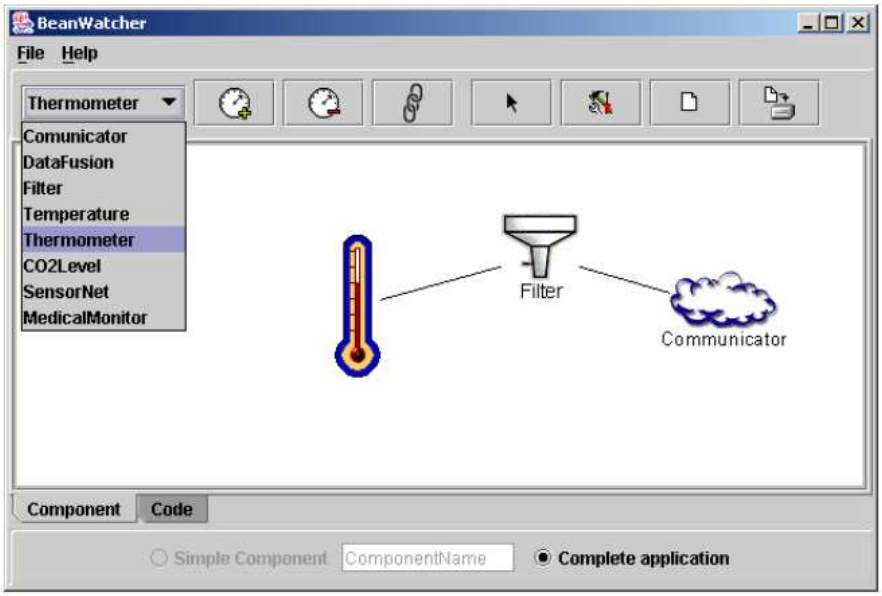


Figura 4.10: Aplicação montada no BeanWatcher.

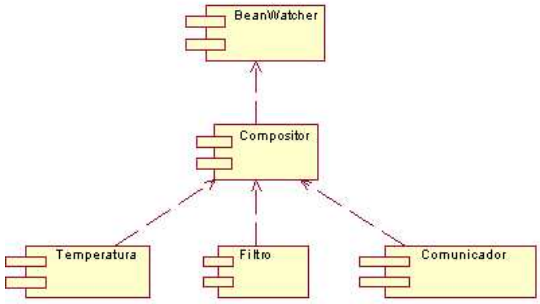


Figura 4.11: Estrutura da aplicação gerada.

4.4 Utilizando o BeanWatcher em um Processo de Desenvolvimento de Software

O BeanWatcher pode ser facilmente utilizado com um processo de desenvolvimento de software. Para exemplificar esta utilização, adotaremos um processo de desenvolvimento genérico como uma simplificação de processos conhecidos, como o Praxis [8] e metodologias orientadas a objetos [5, 24, 18, 32, 2]. Este processo simplificado é composto das seguintes fases:

- **Planejamento**: Nesta fase apenas os requisitos do sistema são identificados; protótipos e a arquitetura das aplicações são especificadas.
- **Construção**: Esta fase é executada em vários ciclos e inclui os passos necessários para o desenvolvimento do sistema, como análise, projeto e implementação.
- **Implantação**: Nesta fase acompanhamos a instalação do sistema para o usuário final, além disso é nesta fase que é feito o treinamento e a documentação final do sistema.

Durante a fase de planejamento, ao especificar os diagramas de caso de uso, podemos identificar os componentes da aplicação a partir dos casos de uso utilizados.

A fase de construção pode ser dividida em quatro passos: análise, projeto, implementação e testes.

- **Análise**: O diagrama de estados pode ser utilizado para modelar o comportamento dos componentes e seus estados durante a execução da aplicação.
- **Projeto**: Devemos desenhar o diagrama de componentes que especifica o relacionamento entre os componentes utilizados pela aplicação. A construção das interfaces do sistema pode ser simplificada uma vez que o BeanWatcher especifica a interface de seus componentes.
- **Implementação**: É feita de forma automática pelo BeanWatcher.
- **Testes**: Todos os testes devem ser implementados, pois a atual versão do BeanWatcher não oferece nenhuma integração com os testes.

A última fase, instalação, inclui documentação técnica e de usuário, são executados todos os testes, treinamento e instalação. Estes fatores não são suportados pelo BeanWatcher e devem ser feitas normalmente.

Como podemos ver o BeanWatcher pode ser utilizado com um processo de forma a otimizar a implementação das aplicações para ambientes sem fio.

Com isso, traçamos três passos que o usuário terá que seguir para desenvolver uma aplicação no BeanWatcher, são eles: construção, implementação e geração (Figura 4.12).

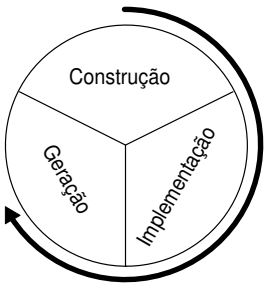


Figura 4.12: Passos para criar uma aplicação com o BeanWatcher.

Na fase construção, o usuário escolhe um ou mais componentes para serem adicionados a área de trabalho, as propriedades dos componentes são informadas e os conectores são inseridos para ligar os componentes. Na fase implementação se for necessário implementar ou otimizar o comportamento de algum componente, utilizaremos a área de edição. Todas as funcionalidades e dependências dos componentes são geradas automaticamente. Na fase geração o usuário solicita que a aplicação seja gerada, podendo assim ser utilizada.

4.5 Considerações Finais

Neste capítulo foi abordado como é o desenvolvimento de aplicações de monitoração e atuação através do BeanWatcher, onde mostramos os requisitos e como uma aplicação

é desenvolvida. Em seguida, mostramos como é a estrutura dos componentes e das aplicações geradas através do BeanWatcher. Por fim, descrevemos a utilização do BeanWatcher em um processo de desenvolvimento de software com o objetivo de mostrar a usabilidade e adequação da nossa ferramenta em ambientes de produção de *software* em maiores escalas, uma vez que para isto sempre é necessário a utilização de um processo.

Capítulo 5

Aplicações Desenvolvidas

Neste capítulo mostraremos algumas aplicações desenvolvidas pelo BeanWatcher para fazer a monitoração remota através de um dispositivo portátil, para isso implementamos o repositório em J2ME.

Para as aplicações mostradas neste capítulo fizemos as seguintes considerações:

- A comunicação é feita com 802.11 (TCP/IP).
- Os dados sensoriados foram simulados por PCs normais, por não possuirmos uma rede de sensores.
- Para o desenvolvimento das aplicações, utilizamos o emulador *J2ME Wireless Toolkit 2.0* [26], disponível pela Sun.
- Na execução das aplicações utilizamos o Palm TungstenC com processador Intel PXA255 de 400MHz, sistema operacional Palm OS 5.2.1, 64 MB de memória e comunicação Wi-Fi 802.11b.

5.1 Aplicação Médica

Nesta aplicação temos um paciente equipado com uma rede de sensores embutida em seu corpo para coletar informações sobre batimentos cardíacos, nível de oxigenação do sangue, temperatura do corpo e pressão arterial. O paciente e o médico com um celular receberiam as informações desta rede de forma que o paciente pode acionar um hospital caso estivesse com algo errado e o médico pode diagnosticá-lo a distância.

Para esta aplicação, precisamos ter uma comunicação em um ambiente para celular, um vez que os dados do paciente seriam mostrados no celular e a visualização dos dados sensoriados devem ser claras e informativas ao médico. Para desenvolver esta aplicação utilizamos os seguinte componentes:

- Componente de monitoração dos batimentos cardíacos e componente de nível de oxigenação do sangue, foram montados com o componente básico *Graph* que representa um gráfico em duas dimensões.
- Componente de pressão arterial, montado com o componente básico *Label* que recebe os dados da pressão e os mostra para o usuário.
- Componente para temperatura corporal, montado com o componente básico *Label* que apenas recebe os dados da temperatura interna do paciente para mostrá-la.

Após a geração dos componentes compostos descritos acima, montamos a aplicação através do BeanWatcher que possuí uma interface com todos os elementos necessários para o médico diagnosticar o problema do paciente. Podemos observar a aplicação executando no Palm TungstenC através da Figura 5.1. Onde temos a monitoração dos dados vitais do paciente.

Figura 5.1: Aplicação médica.

5.2 Agricultura de precisão

Para esta aplicação temos uma plantação equipada com diversos sensores coletando diferentes tipos de dados do ambiente, como temperatura, umidade e acidez do ar ou do solo. Onde um agricultor caminhando recebendo os dados da rede em um dispositivo portátil com interface sem fio.

Precisamos ter uma comunicação em um ambiente *ad-hoc*, um vez que os sensores da plantação não possuem uma estação base para prover a comunicação, a visualização dos dados sensoriados deve ser global e para melhorar o resultado do sensoriamento aplicamos mecanismos de fusão. Para desenvolver esta aplicação utilizamos o componente para rede de sensores e para cada um dos tipos de dados monitorados foi utilizado o componente básico *SensorNet*.

Após isso, montamos a aplicação através do BeanWatcher utilizando para cada dado monitorado o componente funcional *DataFusion*. Esta aplicação pode ser observada executando no Palm TungstenC através da Figura 5.2. Onde em (a) temos o menu para escolhermos que dado queremos ver, em (b) temos a execução da aplicação monitorando a acidez do solo.

5.3 Monitoração ambiental

Nesta aplicação temos o centro de uma cidade equipado com uma rede de sensores para coletar informações sobre a qualidade do ar, para este caso consideremos o nível de gás carbônico. Os cidadãos recebem em seus telefones celulares as informações coletadas

(a) Menu.

(b) Acidez.

Figura 5.2: Agricultura de precisão.

de forma que ele sabem se uma região estar ou não prejudicando sua saúde através dos poluentes existentes.

Para esta aplicação, precisamos ter uma comunicação em um ambiente para celular, uma vez que os dados da qualidade do ar são mostrados no celular. Além disso, a visualização dos dados sensoriados é global, ou seja, deve ser mostrado os dados de uma área monitorada por vários sensores. Para desenvolver esta aplicação utilizamos o componente para gráfico em barras, representado pelo componente básico *BarGraph*.

Esta aplicação pode ser observada executando Palm TungstenC através da Figura 5.3. Onde temos a execução da aplicação para monitorar o nível de gás carbônico de uma região.

Figura 5.3: Aplicação ambiental.

5.4 Monitoração Industrial

Para esta aplicação temos uma fábrica equipada com uma rede sem fio que coleta a temperatura dos maquinários. Um funcionário, utilizando um dispositivo portátil, caminhando pela fábrica recebe os dados dos maquinários através de uma rede sem fio. Além disso, ele pode solicitar que um equipamento desligue, se a temperatura ultrapassar um valor específico.

Consideramos a comunicação em um ambiente *ad-hoc*, um vez que os equipamentos da industria podem não possuir uma estação base para prover a comunicação. Além disso, a visualização dos dados deve ser precisa para que o equipamento possa ser desligado no caso de algum problema. Para desenvolver esta aplicação utilizamos os seguintes componentes:

- Termômetro, para monitorar a temperatura do equipamento remotamente, representado pelo o componente básico *Thermometer*.
- *Led*, para indicar que a temperatura ultrapassou um determinado valor, representado pelo componente básico *LedStop*.
- Alarme, para disparar a *led* ao ocorrer o evento esperado, representado pelo componente funcional *Alarm*.
- Botão, para permitir que o usuário solicite o desligamento do equipamento, representado pelo componente de atuação *SquareButton*.

Após a geração dos componentes descritos acima, montamos a aplicação através do BeanWatcher que possui apenas uma interface com todos os elementos necessários para o usuário visualizar a temperatura do equipamento e solicitar seu desligamento. Esta aplicação pode ser observada executando Palm TungstenC através da Figura 5.4.

Figura 5.4: Aplicação industrial.

5.5 Considerações Finais

Neste capítulo mostramos algumas aplicações desenvolvidas pelo BeanWatcher e que podem ser úteis na área industrial, médica, ambiental, agropecuária entre outras. Como podemos ver o BeanWatcher atende aos requisitos e necessidades inerentes a cada um dos cenários mostrados acima.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Conclusões e Contribuições do Trabalho

Como foi observado no decorrer do trabalho, obtivemos uma ferramenta que permite ao usuário montar suas aplicações de forma simples e rápida, pois através do BeanWatcher, para desenvolver uma aplicação complexa de monitoração em ambientes sem fio ou para redes de sensores basta inserir os componentes desejados, interligá-los e configurá-los, de tal forma que não é preciso escrever código para esta aplicação. Isto permite que usuários, com pouca experiência em programação, possam desenvolver aplicações de forma eficiente.

Ao utilizar o BeanWatcher, o desenvolvedor não está limitado a manipular apenas os componentes disponíveis, ele pode utilizar o *wizard* para inserir novos componentes onde apenas o comportamento deve ser implementado. Isso torna o BeanWatcher extremamente flexível podendo ser utilizado para diversas aplicação.

Outro ponto importante é que as aplicações geradas podem ser utilizadas em qualquer ambiente, pois além de termos o repositório padrão em J2ME, que possui suas próprias características de portabilidade, permitimos a reimplementação dos componentes existentes em outras linguagens, o que torna ainda mais flexível o desenvolvimento através do BeanWatcher.

Com isso, observamos que o BeanWatcher pode ser utilizado em diversas áreas e em diferentes aplicações, listadas a seguir:

- **Fusão de dados**: Apesar de ser utilizados em outras áreas é um importante conceito na área de redes de sensores, pois permite a economia de energia além de auxiliar na extração da informação coletada nos sensores. O BeanWatcher tem suporte a fusão de dados permitindo a utilização de diferentes algoritmos para fusão.

- **Compressão e compactação**: São mecanismos importantes em aplicações para ambientes sem fio, pois diminuem o tamanho das mensagens transmitidas. Caso as aplicações necessitem utilizar estes mecanismos de compressão ou compactação, eles devem ser suportados tanto pelos nós monitorados como pelos nós monitores. O BeanWatcher pode ser utilizado para criar componentes de compactação ou compressão compatíveis com o que esta sendo utilizado nos elementos monitorados.

- **Gerenciamento**: Neste caso o BeanWatcher pode ser utilizado para visualizar alguns elementos de gerenciamento, como mapa de energia ou localização dos nós.

Além disso, podemos desenvolver aplicações onde o gerente pode atuar na rede, como por exemplo solicitar o desligamento de alguns sensores para economizar energia, no caso de redes de sensores.

Além disso, temos a utilização do BeanWatcher em diferentes áreas como:

- **Aplicações médicas**: Para monitoramento remoto de pacientes e diagnósticos através de ambientes sem fio.
- **Aplicações ambientais**: Para monitoração de florestas, rios, mares e qualidade do ar, onde os dados podem ser visto num PC, Palm ou celular.
- **Aplicações industriais**: Para monitorar equipamentos de alto risco ou de difícil acesso onde remotamente o equipamento pode ser desligado.
- **Agropecuária**: Para monitorar grandes plantações podendo atuar na rede solicitando a irrigação.

Dentre as diversas contribuição deste trabalho devemos destacar a utilização de um modelo de componentes que permite a padronização dos elementos monitorados em ambientes sem fio permitindo ao desenvolvedor uma maior reutilização de código.

6.2 Trabalhos Futuros

Como trabalho futuro pode ser desenvolvido suporte à geração de aplicações baseadas em regras, ou seja, se diferentes tipos de dados estão sendo recebidos, a aplicação irá automaticamente escolher a interface para aquele dado de acordo com as regras utilizadas.

Além disso, do ponto de vista das redes de sensores, como utilizamos o PECOS que é destinado para aplicações embutidas, podemos estender o BeanWatcher para gerar aplicações não só para os nós sorvedouros mas também para os nós sensores de forma a termos uma ferramenta que gera aplicações completas para redes de sensores.

Como o BeanWatcher apenas gera código para ser compilado e executado posteriormente, do ponto de vista do repositório em J2ME, poderíamos integrar o BeanWatcher com o emulador para dispositivo portátil de forma a termos a aplicação testada e compilada antes de ser carregado para o Palm.

Poderíamos prover o desenvolvimento de aplicações através dos seus diagramas de componentes em nível de desenho implementável. Para isso utilizaríamos alguma ferramenta de modelagem de sistemas, onde faríamos o diagrama de componentes para em seguida, através do BeanWatcher, a aplicação ser gerada a partir deste diagrama. Com isso, o BeanWatcher poderia ser melhor utilizado por desenvolvedores que utilizam algum processo de desenvolvimento de *software*.

Por fim, poderíamos implementar novos repositórios e prover um mecanismo que permita o desenvolvimento de aplicações utilizando componentes implementados em diferentes linguagens e que suportasse a utilização de aplicações reconfiguráveis utilizando código remoto. Por exemplo, uma aplicação que necessita utilizar um componente que ela não possui então é feita uma solicitação a outro nó e este código ao ser recebido a aplicação seria reconfigurada, dando suporte aos componentes antes não existentes.

Referências Bibliográficas

[1] Mark Baard. Wired news: Making wines finer with wireless. [online] available: http://www.wired.com/news/wireless/0,1382,58312,00.html, access: March 2003, 2003.

[2] Grady Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 2 edition, February 1994.

[3] R. R. Brooks and S. S. Iyengar. *Multi-Sensor Fusion: Fundamentals and Applications*. Prentice Hall, New Jersey, USA, 1998.

[4] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494, September 2002.

[5] Derek Coleman, WithPatrick Arnold, and WithStephanie Bodoff. *Object-Oriented Development: The Fusion Method*. Prentice Hall, September 1993.

[6] Koustuv Dasgupta, Konstantinos Kalpakis, and Parag Namjoshi. Improving the lifetime of sensor networks via intelligent selection of data aggregation trees. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, Orlando, FL, USA, January 2003.

[7] Wagner Luis Cardozo Gomes de Freitas. Algoritmos distribuidos para recuperação de estados em rede de sensores sem fio. Master's thesis, UFMG, Minas Gerais, BR, 2003.

[8] Wilson de Pádua Paula Filho. *Engenharia de Software: Fundamentos, Métodos e Padrões*. LTC, 3 edition, 2003.

[9] H. F. Durrant-Whyte. Sensor models and multisensor integration. *International Journal of Robotics Research*, 7(6):97–113, December 1988.

[10] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *International Conference on Acoustics, Speech, and Signal Processing*, Salt Lake City, USA, June 2001.

[11] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCom'99)*, Seattle, Washington, USA, August 1999. ACM Press.

[12] David Lee Hall. *Mathematical Techniques in Multisensor Data Fusion*. Artech House, Norwood, Massachusetts, USA, April 1992.

[13] Guilherme Campos Hazan. Jsuper waba. [Online] Available: http://www.superwaba.com.br, Access: November 2003.

[14] J. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, pages 12–24, November/December 2002.

[15] IEEE. Wireless standards zone. [online] available: http://standards.ieee.org/wireless/.

[16] National Instruments. Labview 7. [Online] Available: http://www.ni.com/, Access: November 2003.

[17] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom'00)*, pages 56–67, Boston, MA, USA, August 2000. ACM Press.

[18] Ivar Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, June 1992.

[19] D. Johnson, D. Maltz, and Y-C. Hu. The dynamic source routing protocol for mobile ad hoc networks. *Internet-Draft "draft-ietf-manet-dsr-08.txt", Work in progress*.

[20] Bhaskar Krishanamachari, Deborah Estrin, and Stephen Wicker. The impact of data aggregation in wireless sensor networks. In *Proceedings of the International Workshop of Distributed Event Based Systems (DEBS)*, Vienna, Austria, July 2002.

[21] Andre Lins, Eduardo F. Nakamura, Antonio A.F. Loureiro, and Claudionor J.N. Coelho Jr. Beanwatcher: A tool to generate multimedia monitoring applications for wireless sensor networks. In Alan Marshall and Nazim Agoulmine, editors, *Management of Multimedia Networks and Services*, volume 2839 of *Lecture Notes in Computer Science*, pages 128–141, Belfast, Northern Ireland, September 2003. Springer-Verlag Heidelberg.

[22] Andre Lins, Eduardo F. Nakamura, Antonio A.F. Loureiro, and Claudionor J.N. Coelho Jr. Generating monitoring applications for wireless networks. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*, Lisbon, Portugal, September 2003.

[23] Ren C. Luo, Chih-Chen Yih, and Kuo Lan Su. Multisensor fusion and integration: Approaches, applications, and future research directions. *IEEE Sensors Journal*, 2(2):107–119, April 2002.

[24] James Martin and James J. Odell. *Object-Oriented Methods: A Foundation*. Prentice Hall, 2 edition, December 1997.

[25] Sun microsystems. J2me. [Online] Available: http://java.sun.com/, Access: November 2003.

[26] Sun microsystems. J2me wireless toolkit. [Online] Available: http://java.sun.com/, Access: November 2003.

[27] Raquel A. F. Mini, Badri Nath, and Antônio A. F. Loureiro. A probabilistic approach to predict the energy consumption in wireless sensor networks. In *IV Workshop de Comunicação sem Fio e Computação Móvel*, São Paulo, SP, Brazil, October 2002.

[28] Brian Morrissey. 802.11 takes center stage. june, 2002. [online] available:http://www.80211-planet.com/news/article.php/1355221.

[29] Pecosproject. Composition enviroment. [online] available: http://www.pecos-project.org/software.html, access: January 2003.

[30] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[31] Linnyer Beatrys Ruiz, José Marcos Nogueira, and Antônio A. F. Loureiro. Manna: A management architecture for wireless sensor networks. *IEEE Commmunications Magazine*, 41(42):116–125, February 2003.

[32] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorenson. *Object-Oriented Modeling and Design*. Prentice Hall, October 1990.

[33] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, and Mani B. Srivastava. Topology management for sensor networks: Exploiting latency and density. In *2002 ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc'02)*, Lausanne, Switzerland, June 2002.

[34] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman. A taxonomy of wireless microsensor network models, 2002.

[35] Jerry Zhao, Ramesh Govindan, and Deborah Estrin. Residual energy scans for monitoring wireless sensor networks. In *IEEE Wireless Communications and Networking Conference (WCNC'02)*, Orlando, FL, USA, March 2002.

[36] Yonggang Jerry Zhao, Ramesh Govindan, and Deborah Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*, Anchorage, AK, USA, May 2003.