GT-TeI
RT3 - Avaliação dos resultados do protótipo

Silvana Rossetto[1], Bruno Silvestre[2], Noemi Rodriguez[3], Adriano Branco[3], Leonardo K. L. S. Kaplan[3], Ian B. Lopes[3], Douglas V. Santana[2], Vinicius B. da Silva Lima[2], Raul G. M. de Freitas[1] e Denis Takeo Aoki[1]

[1]Departamento de Ciência da Computação (DCC), Instituto de Matemática (IM)
Universidade Federal do Rio de Janeiro (UFRJ)

[2]Instituto de Informática
Universidade Federal de Goiás (UFG)

[3]Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)

8 de agosto de 2014

1. Apresentação do protótipo desenvolvido

O protótipo desenvolvido no escopo deste projeto consiste de: (1) o **testbed de sensores** localizado em um ambiente fechado e constituído de dois tipos distintos de plataformas de sensores; e (2) um **portal Web** para acesso remoto e gerência do testbed de sensores.

O levantamento de requisitos para o projeto deste protótipo foi realizado com base no estudo de outros testbeds (de finalidades similares) e das nossas necessidades como usuários. Entre os testbeds estudados, o WISEBED [WISEBED 2008] e o Motelab (Harvard Sensor Network Testbed) [Werner-Allen et al. 2005] tiveram influência especial.

Nesta seção, descreveremos a arquitetura do testbed de sensores e do portal Web para acesso remoto a esse testbed.

1.1. Arquitetura do testbed de sensores

Um testbed para Redes de Sensores sem Fio (RSSF) é tipicamente composto por um conjunto de nós sensores previamente distribuídos numa área específica. Cada nó da rede é interligado a um computador de controle. O computador de controle, por sua vez, é responsável pela carga do código executável em cada nó e pela interface de interação com o usuário durante a execução dos testes.

A diversidade de plataformas de sensores disponíveis combinada com as diferentes possibilidades de distribuição dos nós de uma rede de sensores possibilitam a construção de testbeds com características bastante distintas. Por exemplo, podemos ter um testbed com os nós distribuídos dentro de uma sala e conectados através de um *hub* USB, ou um testbed com os nós distribuídos em várias salas de um edifício e conectados pela rede Ethernet já existente. Outro exemplo é a fonte de energia dos nós. Podemos utilizar uma fonte local e que depende da existência de uma tomada de energia, ou utilizar o recurso PoE (*Power over Ethernet*) para alimentar o dispositivo via rede Ethernet.

Algumas decisões de projeto, como o tipo de mote (plataforma de sensor) utilizado ou topologia da rede, afetam diretamente os tipos de testes que podem ser executados em

um testbed. Um exemplo é a disposição física dos nós, a qual define o alcance do rádio entre os nós da rede. Podemos ter redes em que todos os nós estão no raio de alcance dos demais nós, ou redes com uma topologia hierárquica e que exigem algoritmos para roteamento de mensagens.

O testbed desenvolvido no escopo deste projeto é voltado para ambientes fechados, sendo constituído por nós sensores independentes e de tipos diferentes, dispostos em uma topologia que pode requerer comunicação multi-saltos, acoplados a um *board* para carga de código e transferência de dados de execução, com fonte de energia constante.

Para atender os requisitos especificados no documento [Rossetto et al. 2014], dividimos a solução final em duas partes. A parte de interface com o usuário, para configuração e monitoração dos testes, é atendida por uma solução de portal Web. A parte responsável pela execução dos testes é atendida por um processo contínuo que monitora quando é necessário executar um teste configurado e dispara as ações necessárias para a rede de sensores do testbed. Essas duas partes são integradas via um banco de dados que armazena as configurações dos testes dos usuários.

A Figura 1 apresenta a arquitetura geral desse testbed, destacando seus módulos principais: kit Nó e kit Servidor.

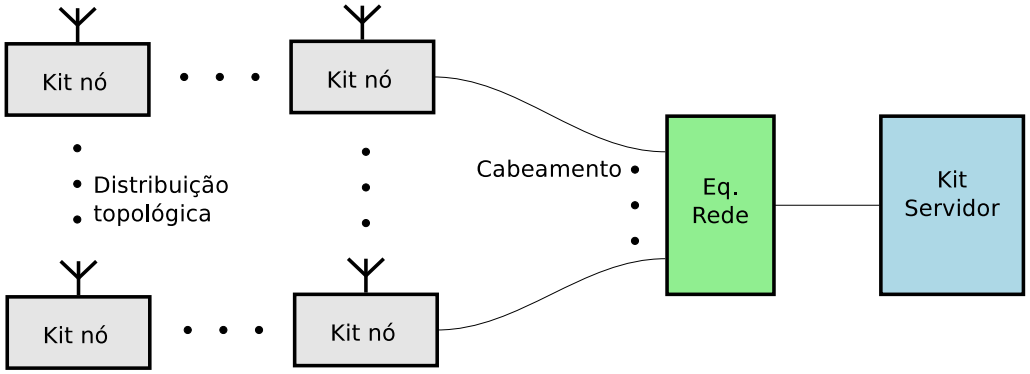

Figura 1. Arquitetura do testbed com seus módulos principais.

O núcleo da arquitetura é o **Kit Nó** que contém os componentes necessários para o funcionamento completo de um nó da rede do testbed: interfaces externas de rádio, sensores físicos, núcleo de processamento, interface de controle e fonte de alimentação de energia). Na Figura 2 apresentamos os elementos constituintes de um Kit Nó.

Cada Kit Nó será ligado, via cabo de rede, a um equipamento de rede (switch) conectado ao Kit Servidor. Essa ligação será usada para a carga das aplicações nos nós sensores (direção Kit Servidor para Kit Nó) e transferência de informações de log de execução das aplicações (direção Kit Nó para Kit Servidor).

O **Kit Servidor** representa um equipamento ou conjunto de equipamentos (servidores e/ou desktops) utilizados no controle central e acesso remoto ao testbed. Este kit é responsável pelas funcionalidades de controle do testbed, servidor de aplicações e banco de dados.

1.2. Arquitetura do portal Web para acesso remoto e gerência do testbed

O portal Web desenvolvido provê acesso remoto e gerência de uso do testbed de sensores.

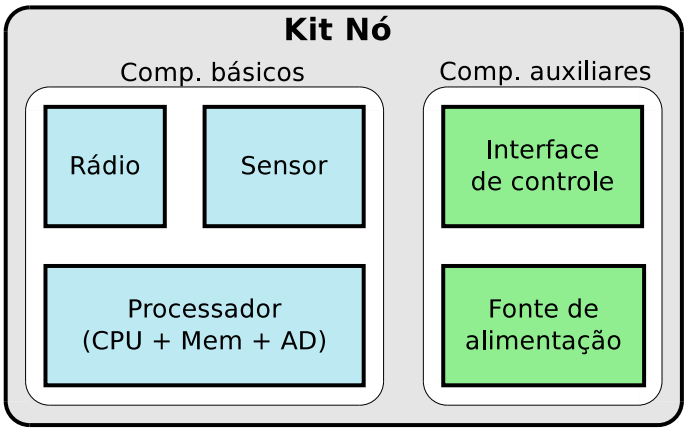

Figura 2. Componentes de um Kit Nó.

Como características adicionais às funcionalidades básicas, destacamos uma arquitetura que permite a configuração do sistema para reutilização em diferentes testbeds, incluindo testbeds heterogêneos compostos por diferentes tipos de nós sensores e diferentes topologias de rede. Dessa forma, o Sistema de Controle proposto pode ser utilizado tanto em testbeds de uso geral como em testbeds com finalidades mais específicas. Também pode ser configurado para topologias de rede simples ou topologias mais complexas, como por exemplo, um testbed distribuído num prédio de vários andares.

O testbed é formado basicamente por três partes: a rede de sensores sem fio; o controle geral; e a interface de operação. A Figura 3 apresenta os principais componentes do sistema.

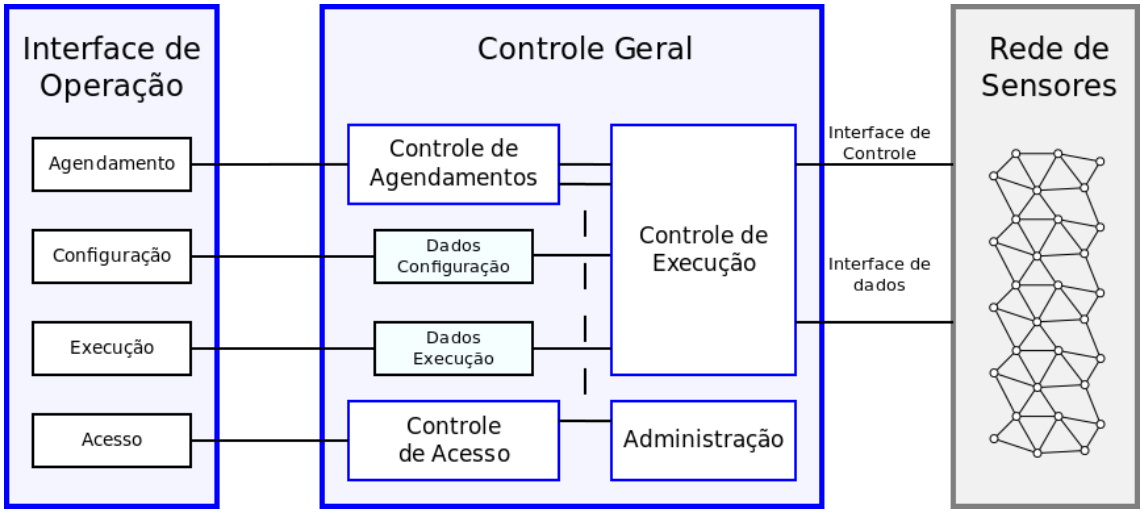

Figura 3. Arquitetura funcional do sistema Testbed

A **rede de sensores sem fio** é formada pelo conjunto de dispositivos distribuídos fisicamente por uma área que forma o testbed. A comunicação entre os nós é feita via rádio e o alcance entre cada par de nós da rede depende da distância entre eles e da potência do sinal de rádio. No caso específico do nosso testbed, todos os nós da rede estão acoplados fisicamente a um dispositivo conectado a uma rede cabeada. Essa conexão serve tanto para o controle da carga de código executável nos nós sensores, quanto para o

envio de mensagens de dados pelo canal serial de cada nó.

O **controle geral** é composto por alguns módulos básicos que controlam a execução dos testes no testbed. As principais funcionalidades são o Controle de Execução, o Controle de Agendamentos das janelas de execução e o Controle de Acesso.

A **interface de operação** é composta por quatro funcionalidades: Agendamento, Configuração, Execução e Controle de Acesso. A Interface de Agendamento permite ao usuário consultar, reservar e cancelar as janelas de execução no testbed. Na Interface de Configuração o usuário pode configurar a topologia da rede com a ativação e desativação de determinados nós, armazenar o seu código executável para posterior execução na rede de sensores e definir condições de execução específicas para o seu teste. A Interface de Execução permite ao usuário controlar e monitorar o andamento dos testes. A Interface de Acesso controla o acesso do usuário aos recursos do testbed.

A Figura 4 apresenta a arquitetura do sistema com uma visão geral dos módulos e das tecnologias utilizadas.

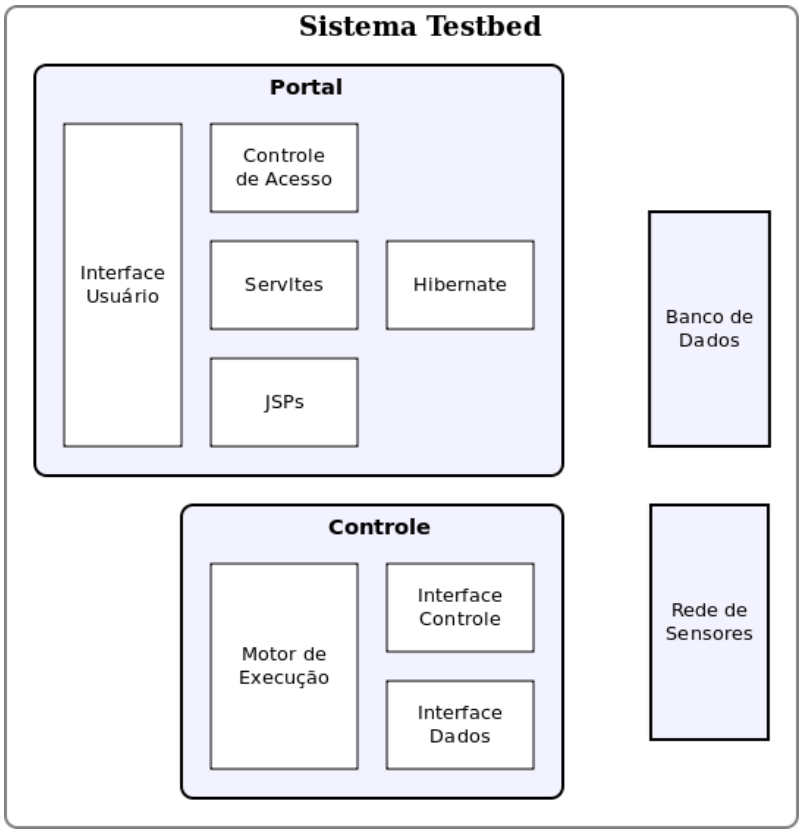

Figura 4. Arquitetura Geral

1.2.1. Módulos do Sistema – Portal

O funcionamento geral do portal consiste em primeiro verificar se o usuário está logado no sistema (Controle de Acesso) e, caso esteja, permitir a ele fazer uma requisição via página `html` (Interface Usuário). Os `servltes` capturam e processam essa requisição e utilizam o `hibernate` para fazer acesso ao banco. Feito isso, os `servltes` redire-

cionam o fluxo com os dados para uma página `jsp` que por fim irá gerar a página `html` visualizada pelo usuário.

As telas do Portal permitem ao usuário configurar no banco de dados os testes que o Controle irá executar. Durante a execução do teste, o controle do sistema armazena o `log` no banco de dados, permitindo a leitura por parte do portal. Isso possibilita ao usuário ter acesso online aos dados do seu teste em execução.

O projeto do portal seguiu o padrão de arquitetura `Model-View-Controller` (MVC).

A `View` contém as páginas `jsp` e `html`, sendo localizadas no diretório `WebContent` do projeto.

O `Controller` possui todos as `classes java` e os `servlets` responsáveis pela parte de mapeamento e controle das ações. Trata-se da parte intermediária entre uma solicitação do usuário e o atendimento do sistema. Suas classes estão definidas no pacote "controle".

O `Model` é responsável pela manipulação dos dados no banco, fornecendo as funcionalidades para o `Controller` acessar as informações. Os pacotes do projeto "dao" e"model" possuem os códigos `java` com essas funcionalidades.

View As páginas foram implementadas utilizando `jsp`. Esta tecnologia permite que se insira código `java` no desenvolvimento da página. Essa característica é fundamental para manipular os dados retornados pelo controle.

O `javascript` foi utilizado para cuidar da parte dinâmica das telas, tanto a parte gráfica quanto as requisições de controle. A tecnologia utilizada para realizar estas requisições foi o AJAX. Também foi utilizado `javascript` na validação dos campos de entrada de dados. As requisições apenas são repassadas ao servidor caso os campos estejam corretamente preenchidos. No caso de um campo inválido, uma janela *pop-up* é invocada informando ao usuário qual foi o erro cometido.

Controller O `controller` utiliza `servlets` e classes `java`. Os `servlets` são utilizados para capturar os formulários preenchidos pelo usuário, manipular os dados e retornar o fluxo para a tela correspondente. As classes `java` são utilizadas quando ocorre a necessidade de realizar determinada funcionalidade na própria página a ser carregada. De forma geral, os `servlets` tratam as requisições do usuário e as classes `java` retornam dados para a própria página.

Model A ferramenta utilizada para realizar acesso ao banco foi o `hibernate`. Os pacotes "dao" e "model" possuem as classes necessárias para que o `hibernate` consiga acessar e mapear o banco. O pacote "dao" é responsável pelas ações diretas no banco (leitura, armazenamento e consulta) e o pacote "model" contém o mapeamento do banco em classes para que o acesso seja feito através de objetos. As classes com essa funcionalidade são chamados de `Beans`. Nelas estão contidas as operações `get/set` de cada elemento. Cada instância do banco se torna um objeto que é repassado ao `Controller`.

Acesso ao banco de dados Cada tabela do banco de dados é mapeada pelo `hibernate` para uma classe `java`. Essas classes são utilizadas nas leituras, escritas e consultas ao banco.

O ciclo básico de utilização consiste em criar um objeto (`bean`) do tipo da tabela que se deseja fazer uma operação e utilizar uma função que execute tal operação (de um classe do pacote "dao"). Caso haja retorno (consulta ou leitura) será retornado um objeto do tipo da tabela acessada.

Cada tabela do banco possui sua própria classe no pacote "dao". Basicamente, cada uma dessas classes possue as funções de persistir, apagar e retornar um objeto, além de listar todos elementos de determinada tabela.

As `procedures` definidas no banco também são invocadas pelo portal através do `hibernate`. Abre-se uma conexão entre o portal e o banco de dados. O código da `procedure` é repassado para o banco e o resultado é retornado ao portal como uma lista.

Upload de Arquivos O framwork utilizado para auxiliar nessa tarefa foi o "UploadBean". Este framework faz um papel de intermediário entre uma página `html`, que informa o arquivo, e o banco de dados, que armazena os dados.

O "UploadBean" é um `servlet` parametrizável que faz parte do `Controller`. Ele foi parametrizado para armazenar os dados dos arquivos na tabela "userfiles" do banco de dados.

Login O módulo de login controla as permissões dos usuários, i.e., verifica se o usuário logado está autorizado ou não para acessar determinada página.

A estrutura de login funciona da seguinte maneira. O usuário informa seu *login* e sua senha. A API JAAS verifica no banco de dados se esse usuário existe e se possui permissão de acessar a página solicitada. Em caso positivo, libera o conteúdo, em caso negativo informa que o usuário não existe ou que o conteúdo solicitado não é permitido para o nível de acesso do usuário.

O JAAS restringe as permissões baseado no conceito de tipo de usuário. Para este sistema foram criados três tipos de usuários. Aluno, Professor e Administrador. Usuários do tipo Aluno e Professor podem acessar todas as páginas, exceto as de administração, que são apenas permitidas para o tipo Administrador.

A fonte de dados utilizada para reconhecer os usuários foi o próprio banco de dados do sistema. As informações de *login* e senha estão localizadas na tabela "users" com os atributos "ulogin" e "upassword", respectivamente. Os tipos de usuários estão localizados na tabela `user_roles` e seu atributo é `role_name`.

A definição de quais páginas são permitidas para determinado tipo de usuário está no arquivo `web.xml`. A configuração da conexão com o banco de dados é feita no arquivo `context.xml`.

Apesar das informações de usuários serem coletadas do próprio banco de dados do sistema, a mudança para outra fonte é possível. Caso se deseje obter as informações de usuários de um servidor LDAP, por exemplo, basta configurar o arquivo `context.xml`

devidamente para que o JAAS faça a conexão com o próprio servidor LDAP, ao invés do banco de dados do sistema. Nesse caso será necessário implementar uma modificação no portal para que o sistema de *login* crie automaticamente na tabela local os novos usuários LDAP.

1.2.2. Módulos do Sistema – Controle

A Controle do testbed é responsável pela execução dos testes configurados no banco de dados. Dividimos o controle em três módulos.

Motor de Execução Trata-se do principal módulo do controle do testbed sendo responsável pela execução dos testes agendados. Controla o início e o fim de cada teste e executa os comandos de carga de código binário, de acordo com a sequência de passos do script de teste do usuário. Também permite o controle remoto do experimento com as funcionalidades de pausa e reinício de um teste do usuário.

Módulo construído em Java, utiliza o padrão de projeto *Singleton* para centralizar serviços como, por exemplo, uma fila de eventos, e o acesso ao banco de dados. Acessa o banco de dados via JDBC-Driver do PostgreSQL. O controle remoto é implementado como um serviço TCP/IP e utiliza um protocolo específico. A mensagem do protocolo é uma string de texto composta por dois números inteiros separados por espaço. O primeiro número indica o *testid* corrente e o segundo indica um comando. O comando 0 e 1 representam, respectivamente, a solicitação de início e a de parada. O comando 2 indica que um teste foi inserido ou atualizado no banco de dados. O Controle só executa o comando se *testid* for o mesmo do teste que está sendo executado. Essa restrição garante que somente as aplicações com permissão de acesso ao banco de dados, como o portal, podem recuperar o *testid* corrente e efetivar o comando.

Os parâmetros para acesso ao banco de dados e a configuração do serviço TCP/IP são definidos por um arquivo de configuração.

Interface de Controle O usuário deve compilar o seu código fonte na sua máquina local e depois transferir esse código compilado para o servidor via interface do portal. Então o Controle executa o procedimento de conversão e carga do código nos respectivos nós da rede.

No caso do usuário utilizar o framework TinyOS, ele deve utilizar os seguintes comandos no processo de compilação na sua máquina local: *make micaz* ou *make telosb*. Esses comandos geram um arquivo *main.exe* nos respectivos diretórios *build/micaz* ou *build/telosb*. São esses arquivos que devem ser transferidos para a máquina do servidor via interface do portal.

O Controle executa os comandos de carga de código binário nos nós da rede. Ele é composto pela ferramenta que atualiza, dentro do código executável, o ID lógico do nó, e também pelas ferramentas específicas de transferência do código binário para cada tipo de nó sensor utilizado na rede.

O procedimento de conversão dos arquivos utiliza os seguintes programas:

- Para MicaZ: *avr-objcopy* e *tos-set-symbols*
- Para TelosB: *msp430-objcopy* e *tos-set-symbols*

O procedimento de carga de binário utiliza os seguinte programa:

- Para MicaZ: *uisp*
- Para TelosB: *tos-bsl*

Interface de Dados Controla o redirecionamento das interfaces de dados de cada nó para as portas TCP/IP na máquina servidora do Testbed. Com isso as portas de dados podem ficar disponíveis no endereço da máquina servidora. O número da porta de acesso de cada nó é obtido pela soma de um valor base com o ID físico do nó. O valor base é configurado no arquivo de parâmetros conforme, definido na subseção 1.2.5. A nossa instalação default assume que o valor base é 10000. Dessa forma a faixa de acesso, se considerarmos 10 nós, ficará de 10001-10010.

1.2.3. Modelo de dados

Na Figura 5 apresentamos o diagrama do modelo de dados do sistema. As linhas de relacionamento cheias representam relações entre campos chaves, enquanto que a linhas pontilhadas representam relacionamentos normais. A seguir descrevemos as principais tabelas

Podemos dividir o modelo em sete partes:

- Controle do usuário
  - *users* - Cadastro dos usuário do sistema.
  - *userroles* - Cadastro dos papeis (roles) do usuário. Usado nas permissões de acesso a cada tela do portal.
- Configuração da rede de sensores sem fio
  - *network* - Identifica as características e parâmetros para cada nó sensor da rede.
  - *nodestype* - Identifica os tipos de nós utilizados na rede.
- Registro de logs
  - *logdata* - Armazena os logs do sistema.
- Controle de Agenda
  - *agenda* - Armazena as agendas dos usuários.
  - *slotstemplate* - Configuração padrão (semanal) da disponibilidade de slots por tipo de nível de acesso.
  - *extraslot* - Reprogramação eventual dos slots para uma determinada data/hora.
  - *slotsviewtype* - Tabela com a definição dos campos da "View" de busca de slots.
- Configuração dos Nós da rede
  - *netconfig/netconfignodes* - Armazena os detalhes das configurações personalizadas dos nós da rede para uso nos testes do usuário.
  - *netconfigdefaultfiles* - Identifica os arquivos default a serem carregados nos nós da rede para uma determinada configuração.
  - *userfiles* - Armazena os arquivos binários para serem utilizados nos testes.
- Plano de execução

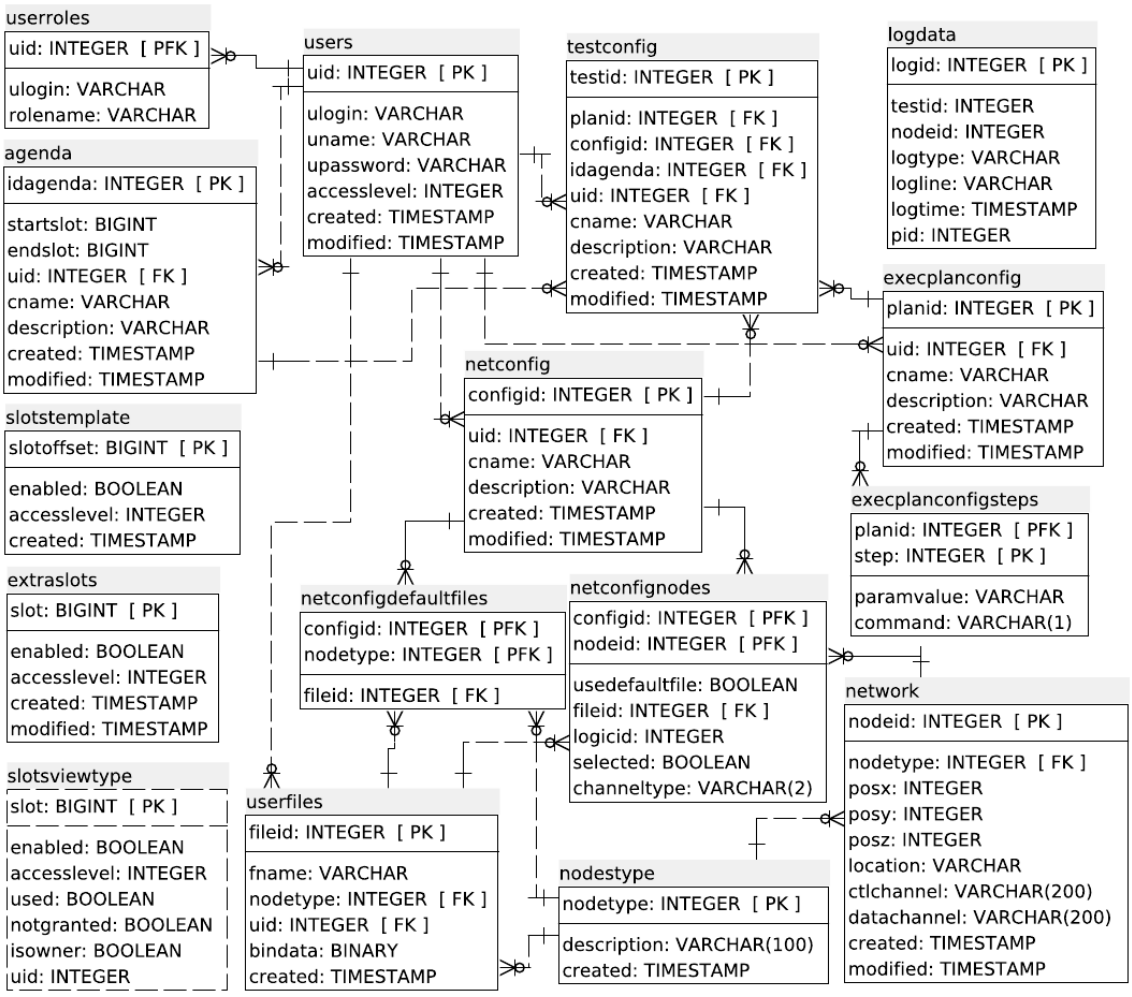

Figura 5. Modelo de dados

  - *execplanconfig/execplanconfigsteps* - Armazena os passos de execução para uso nos testes do usuário.
- Confirmação de um teste
  - *testconfig* - armazena a confirmação da configuração de cada teste agendado.

1.2.4. Descrição do sistema – Portal Testbed

Os módulos do portal tesbed incluem as seguintes funcionalidades: Administração, Agenda, Topologia, Script de Execução, Confirmação do Teste e Monitoração.

Administração Essa tela permite o controle das contas de usuário. Nela é possível comandar as seguintes operações para cada usuário:

- Adicionar;
- Editar/Atualizar;
- Deletar.

O acesso a essa janela só é permitido aos usuários cadastrados como administradores do sistema.

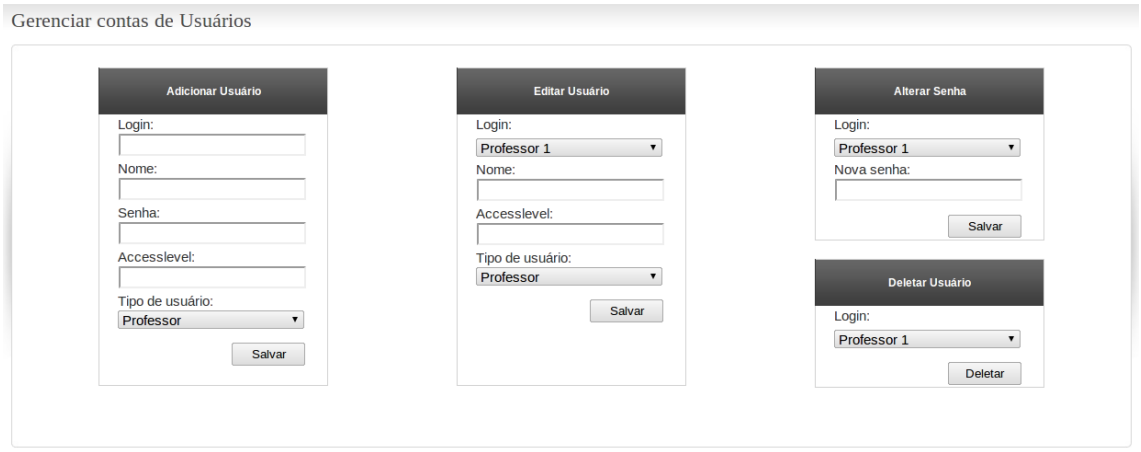

Figura 6. Tela de configuração das contas de usuários (Administração)

Descrição da Interface A interface foi dividida em 4 *tables* intituladas de *Adicionar Usuário*, *Editar Usuário*, *Alterar Senha* e *Deletar Usuário*.

Para adicionar um usuário deve ser informado um Login, Nome, Senha, Accesslevel e o tipo de usuário na *table* intitulada *Adicionar Usuário*.

A edição de usuários permite alterar o nome, accesslevel e o tipo de usuário. Na *listbox* do campo Login, da *table Editar Usuário*, são mostrados todos as contas de usuários que o sistema possui.

A *table Alterar Senha* possui dois campos. Um para selecionar a conta que se deseja alterar a senha e outro campo para informar a nova senha, além de um botão de confirmação dessa ação.

O último campo é o de deletar usuário. Nele há o campo para escolher a conta a ser deletada e o botão para acionar essa ação. Todas as informações contidas no sistema referente ao usuário serão apagadas, isto é, reservas, arquivos, topologias e planos de execução.

Principais classes

- Usuario_down
  - Adiciona uma conta de usuário ao banco
  - Parâmetros
    - Login - entrada - Login do usuário
    - Nome - entrada - Nome do usuário
    - Senha - entrada - Senha do usuário
    - Accesslevel - entrada - Accesslevel do usuário
    - Tipo - entrada - Tipo de conta do usuário
- Usuario_edit
  - Edita conta de usuário

    - Parâmetros
        * Login - entrada - Login do usuário
        * Nome - entrada - Novo nome do usuário
        * Accesslevel - entrada - Novo accesslevel do usuário
        * Tipo - entrada - Novo tipo de conta do usuário
- Usuario_setsenha
    - Altera senha da conta do usuário
    - Parâmetros
        * Login - entrada - Login do usuário
        * Senha - entrada - Nova senha do usuário
- Usuario_deleta
    - Apaga conta e configurações do usuário
    - Parâmetros
        * Login - entrada - Login do usuário

Agenda A tela de agenda possui a finalidade de controlar as reservas feitas para utilização do testbed. Disponibiliza as seguintes funcionalidades:

- Consultar horários;
- Realizar uma reserva;
- Verificar reservas;
- Cancelar reserva.

O processo básico para consultar horários disponíveis requer apenas que o usuário informe qual é a data desejada. A reserva é feita de forma similar. Escolhe-se a data, depois o horário e a quantidade de tempo que se deseja utilizar o testbed. Para verificar reservas já realizadas, basta selecionar uma das reservas disponíveis. Por último, para cancelar uma reserva, basta informar a data da reserva, selecioná-la e cancelá-la.

Descrição da Interface A interface possui um campo onde o usuário pode escolher a data. Neste campo, pode-se escolher por dia, mês ou ano, através das respectivas *listbox*. Também é possível escolher a data através do calendário disponibilizado ao lado desses *listbox*. O calendário aparece na tela após clicar no seu ícone.

Ao ser selecionada a data, um `servlet` invoca uma `procedure` no banco que retorna os horários disponíveis. De posse desses horários, a *table* da agenda é preenchida.

A *table Realizar Reserva* possui a função de atribuir um nome, uma descrição (opcional) e a quantidade de horas que se deseja utilizar uma reserva.

As consultas são realizadas pela *table Consultar Reservas*. Nela são mostrados as informações de reserva, isto é, o nome, dia, horário, duração e descrição.

O cancelamento é feito na *table Cancelar Reservas*.

Principais classes

- Agenda_get
    - Retorna horários da agenda
    - Parâmetros

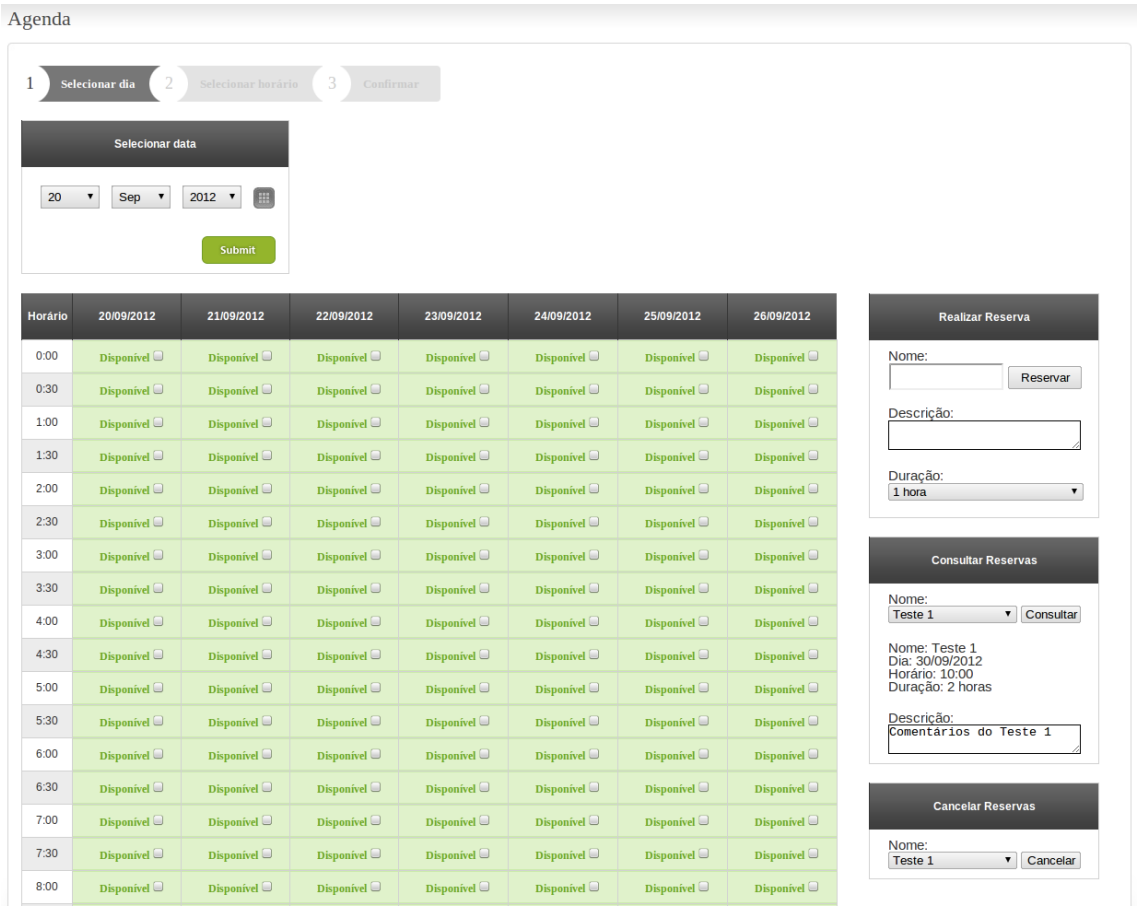

Figura 7. Tela da agenda do testbed (Agenda)

    - Data - entrada - Data a ser consultada os horários
    - List - retorno - Lista com os horários
- Agenda_down
  - Armazena agenda no banco
  - Parâmetros
    - Data - entrada - Data a ser criado o horário
    - StartSlot - entrada - Horário inicial da reserva
    - EndSlot - entrada - Horário final da reserva

Topologia A tela para configuração da topologia possui a finalidade geral de informar as configurações específicas para cada mote que irá participar do teste. A tela de *Topologia* disponibiliza as seguintes configurações para cada mote do Testbed:

- Escolha de um ID lógico;
- Inclusão ou exclusão do respectivo nó no teste;
- Seleção do arquivo a ser carregado no próprio sensor;

Para facilitar a configuração em lote foram utilizadas quatro estratégias. A primeira é o pre-estabelecimento do ID lógico como sendo igual ao do ID físico, visto que para a maioria dos testes é uma situação comum. A segunda estratégia foi a criação de um elemento que torne possível, com um clique, incluir ou excluir todos os sensores de um

teste. A terceira estratégia foi o estabelecimento de arquivos default associados aos tipos de motes. Dessa forma, não é necessário selecionar um arquivo para cada mote e sim um arquivo para cada tipo de mote. Essa estratégia levou em consideração o fato que na maioria dos testes utiliza-se o mesmo arquivo em todos os motes, com exceção do mote BaseStation. A quarta e última estratégia foi o pre-estabelecimento do canal de dados desligado para todos os motes, também visto que na maioria dos casos somente utiliza-se o canal de dados para o mote BaseStation.

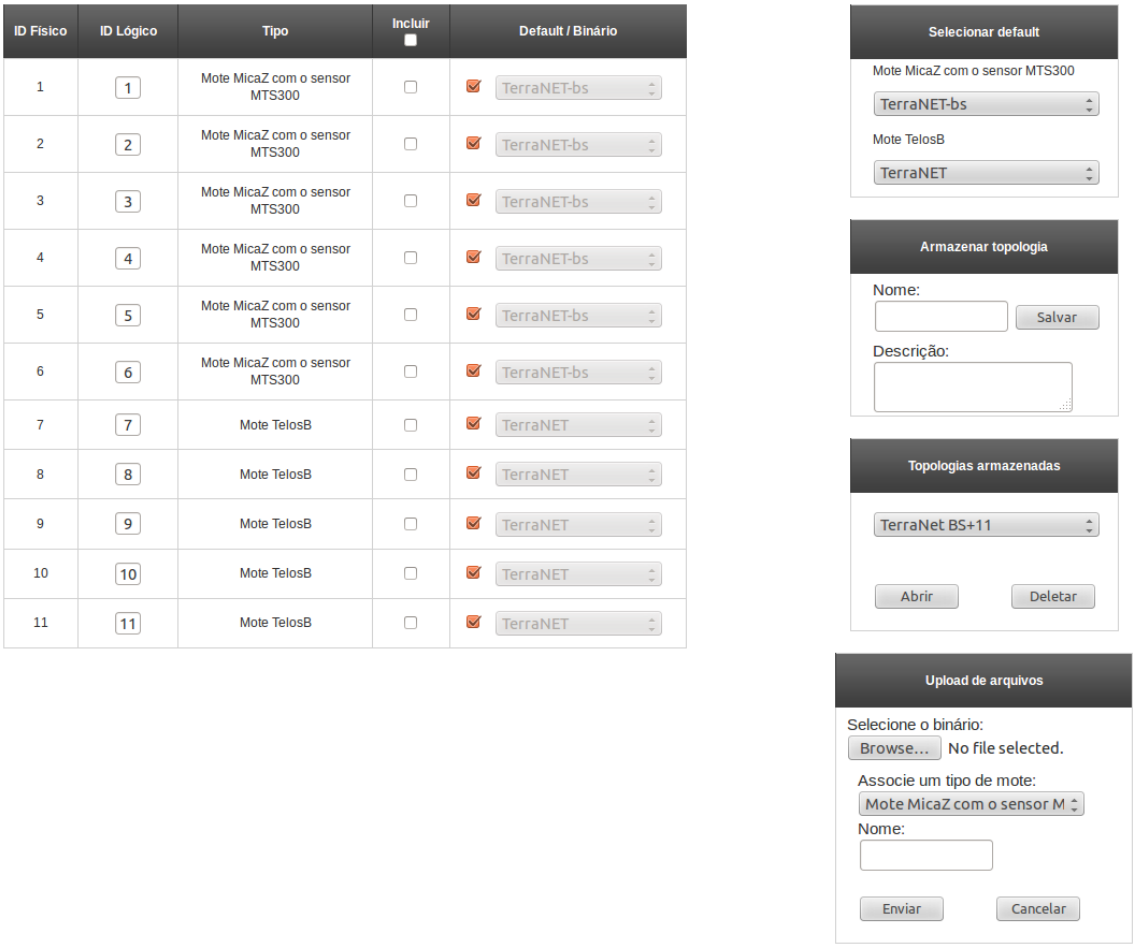

Figura 8. Tela de configuração da Rede (Topologia)

Descrição da interface A interface foi dividida em duas partes conforme a figura 8. No lado esquerdo encontra-se uma *table* que permite a configuração dos motes, e no lado direito estão localizados quatro *tables* alinhadas verticalmente. Estas permitem operações de carga e armazenamento de topologias, carga de um arquivo binário e identificação dos arquivos default.

A *table* que configura os motes é gerada dinamicamente a partir das informações encontradas no banco. A descrição e o tipo do mote são consultados da tabela *nodestype*. O ID físico e a quantidade total de motes são fornecidos pela tabela *network*. Depois de obtidas estas informações, a tabela é gerada. Essa característica dinâmica aumenta a flexibilidade do portal, afinal caso haja substituição, remoção ou adição de motes no

testbed, o sistema não será invalidado, desde que os novos valores sejam corretamente atualizados no banco.

Essa *table* consiste de seis colunas que são: o *ID Físico, ID Lógico, Tipo, Incluir e Default/Binário*. Cada linha representa um mote do testbed.

O *ID Físico* é fixo não podendo ser alterado pelo usuário, pois está associado ao identificador do nó na rede física.

A coluna *ID Lógico* representa o *ID* de endereçamento do nó na rede e recebe como valor default o *ID Físico*. O usuário pode alterar esse identificador para qualquer valor inteiro positivo de 16 bits. Se for necessário, é permitido que haja repetição de valores.

O *checkbox* marcado na coluna *incluir* sinaliza que o respectivo mote será incluído na execução do teste. Uma facilidade foi fornecida neste campo. Ao marcar o *checkbox* do cabeçalho da tabela em *incluir*, todos os motes serão marcados para participar do teste. Ao desmarcá-lo, todos serão excluídos. Isso evita que o usuário tenha que selecionar linha por linha para incluir ou excluir todos motes em seu teste.

Na coluna seguinte, *Default/Binário*, ocorre a seleção (ou não) para o mote utilizar o arquivo estabelecido como default para o teste. A marcação do *checkbox* implica na seleção do arquivo default, impossibilitando a escolha de qualquer outro arquivo. A desmarcação da mesma *checkbox* viabiliza a seleção de algum arquivo previamente carregado no portal.

Na *table* intitulada *Selecionar default* serão listados todos os tipos de motes que o testbed possui. O sistema consulta a tabela *nodestype* e imprime na tela, para cada mote encontrado, a descrição e as possíveis seleções de arquivos que o usuário possui. A busca desses dados é feita consultando-se a tabela *userfiles*. Desta forma, são apresentados apenas os arquivos para seu respectivo mote e que pertençam ao próprio usuário. Caso o usuário não tenha carregado nenhum binário para um determinado tipo de mote, ele será informado que nenhum arquivo dele está associado para aquele respectivo mote. É necessário que, para cada tipo de mote, seja definido um arquivo default.

Para armazenar a topologia, o sistema utiliza a *table* intitulada *Armazenar topologia*. Nela é obrigatório a escolha de um nome, mas não de uma descrição.

O usuário pode abrir ou deletar uma topologia previamente armazenada através da *table* intitulada *Topologias armazenadas*. Na *listbox* são mostradas todas as topologias que o banco possui, cujo proprietário seja o próprio usuário. Selecionando uma topologia, uma requisição é enviada para a própria página que preenche a tabela de configuração conforme a topologia carregada. O botão *Deletar* apaga a topologia selecionada do banco de dados.

A última funcionalidade da tela de topologia é a carga de um arquivo. É possível carregar um arquivo da máquina do cliente para máquina do servidor. Concluído o processo, a página é recarregada, agora já dispondo do arquivo armazenado. O botão *Cancelar* apaga uma seleção prévia de algum arquivo.

Principais classes

- Topologia_get
  - Retorna todas as topologias que o usuário possui armazenadas no banco de dados
  - Parâmetros
    - UserID - entrada - Identificador do usuário
    - List - retorno - Lista com as topologias
- Topologia_down
  - Servlet que armazena as configurações de topologia feita pelo usuário
  - Parâmetros
    - Request - entrada - Possui todos as informações da topologia
- Topologia_deleta
  - Servlet que deleta determinada topologia
  - Parâmetros
    - Request - entrada - Possui todos as informações da topologia

Melhorias futuras As seguintes melhorias foram identificadas:

- Caso o usuário não utilize todos os tipos de motes em seu teste, permitir a configuração do teste sem necessitar associar para cada tipo de mote um arquivo default;
- Limitar o campo da coluna *ID Lógico* apenas a dígitos numéricos.

Script de Execução O script de execução consiste em informar ao sistema uma determinada sequência de passos e intervalos para ativar ou desativar os nós sensores. Ao todo três funcionalidades são fornecidas ao usuário:

- Área para que seja detalhado o script de execução;
- Operação para armazenar o script de execução no banco de dados;
- Operação para carregar ou deletar um script de execução previamente armazenado.

Além dessas funcionalidades, o portal descreve como é a sintaxe dos comandos do script, ou seja, a forma padrão de como o usuário deve informar os comandos e respectivos parâmetros.

Figura 9. Tela Script de Execução

Descrição da interface A tela apresentada na Figura 9 possui quatro *tables* ao todo, divididas em três colunas. A primeira *table*, intitulada de *Descrição da sintaxe*, descreve a sintaxe da linguagem de script definida. A *textarea* ao lado é onde o usuário deve digitar o script de execução conforme a sintaxe descrita. As duas *tables* localizadas mais a direita são para armazenar, carregar ou apagar um script de execução do banco de dados.

A *textarea* recebe as informações das operações que serão realizadas. Permite-se que o usuário digite qualquer tecla, sendo feita uma verificação sintática apenas no processo de armazenamento.

Os operadores 'A' e 'D' (Activation/Deactivation) devem ser seguidos por números positivos separados por vírgulas, pela palavra 'ALL' ou por uma indicação de faixa de valores indicando o início e o fim. O operador 'W' (Wait) deve ser seguido de um único número positivo.

Para salvar o script de execução, o usuário deve informar um nome e uma descrição opcional. Feito isso, basta clicar em salvar. O script de execução a ser armazenado terá sua sintaxe avaliada e, caso esteja correta, será armazenado no banco de dados. Após esse processo a tela é recarregada e disponibilizado ao usuário com seu script salvo.

Também é possível carregar um script de execução, do próprio usuário, para a *textarea* a fim de editá-lo. Os scripts disponíveis são apresentados na *listbox* da *table* intitulada *Script de execução armazenados*. Será feita uma busca no banco de dados que retornará para a *textarea* o Script carregado, sendo possível a sua edição. Nessa mesma *table* é possível fazer a deleção de determinado script.

Sintaxe

- Ativar um nó:
  - Comando: A ALL| < id > | < idList > | < idRange >
- Desativar um nó:
  - Comando: D ALL| < id > | < idList > | < idRange >
- Esperar execução:
  - Comando: W < time >
- Definição dos parâmetros:
  - ALL: Constante que define todos os nós selecionados na configuração
  - < id >: número inteiro
  - < idList >: sequência de < id > separados por vírgula
  - < idRange >: < idA > ... < idB >, onde seleciona todos os ids de idA até idB
  - < time >: quantidade de segundos

Exemplos de scripts

- Os seguintes scripts são válidos:
  - Exemplo 1
    A 1,2,3
    D 2,3,5
    w 2

  - Exemplo 2
    d 1,2
    w 1
    A 1
- Os scripts não são válidos:
  - Exemplo 3
    A 2, 3, 4
  - Exemplo 4
    w 2,3
  - Exemplo 5
    A
    D 2,3

O exemplo 3 não é aceito por conta do espaçamento entre os atributos. O exemplo 4, por incluir dois atributos no operador *w*. E o exemplo 5 não é aceito por não possuir parâmetros após o operador *A*.

Principais classes

- ScriptExecucao_down
  - Servlet que faz a análise sintática do script e o armazena no banco de dados
  - Parâmetros
    - Request - entrada - Possui o script informado pelo usuário
- ScriptExecucao_deleta
  - Servlet que apaga determinado script de execução
  - Parâmetros
    - Request - entrada - Possui o identificador do script a ser apagado

Trabalhos futuros As seguintes melhorias podem ser realizadas futuramente:

- Realizar a verificação sintática no lado cliente;
- Permitir separar os atributos do script de execução por espaços em branco.

Confirmação do Teste Responsável por fazer a conexão entre uma reserva, uma topologia e um plano de execução para a criação de um teste.

Para atingir seu objetivo, essa tela lista todas as reservas, todas as topologias e todos os planos de execução que o usuário possui. Feito isso, o usuário pode fazer uma conexão entre esses itens e assim confirmar um teste. Os parâmetros são gravados no banco na tabela *testconfig*.

Descrição da interface A interface é constituída de uma única *table* que lista, em *listboxs* separadas, todas as as reservas, topologias e planos de execução do usuário armazenados no banco de dados. Deve-se atribuir um nome ao teste e opcionalmente uma descrição.

O usuário deve possuir ao menos uma reserva, uma topologia e um plano de execução para a confirmação de um teste. Satisfeita essa condição, são listados os respectivos atributos nos *listboxs* com os campos agenda, topologia e plano de execução, cabendo ao usuário escolher dentre elas aquelas que serão utilizadas em seu futuro teste.

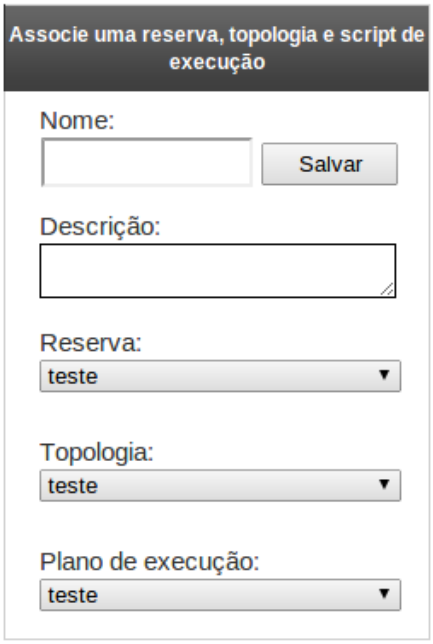

Figura 10. Tela de Confirmação do Teste

Principal classe

- TestConfig_down
  - Servlet associa reserva, topologia e plano de execução para criar um teste
  - Parâmetros
    - Request - entrada — Possui o identificador do usuário, agenda, topologia e plano de execução

Monitoração A sessão de monitoração tem o propósito de informar ao usuário todos os dados de seu teste corrente. Consiste de uma única janela nomeada de *Monitoração*. Esta tela fornece:

- Os dados de teste do usuário;
- O log do teste corrente;
- As operações para iniciar ou parar uma execução corrente.

O portal avisa caso não exista um teste corrente para o usuário.

Descrição da interface Na Figura 11 apresentamos a tela de Monitoração. A tela consiste de dois botões localizados na parte superior — *start* e *stop* — e de uma área que mostra o log do teste corrente.

O botão de *start* envia um sinal para o Controle do Testbed que por sua vez reinicia o teste. O botão de *stop* funciona de forma similar, parando o teste corrente. Para ambos, uma mensagem é mostrada logo abaixo, informando ao usuário se o controle obteve sucesso ou não ao executar a ação solicitada.

O log é mostrado em uma parte delimitada da tela. O usuário pode navegar verticalmente ou horizontalmente através dos *scrolls* laterais. Em cada linha é mostrado o

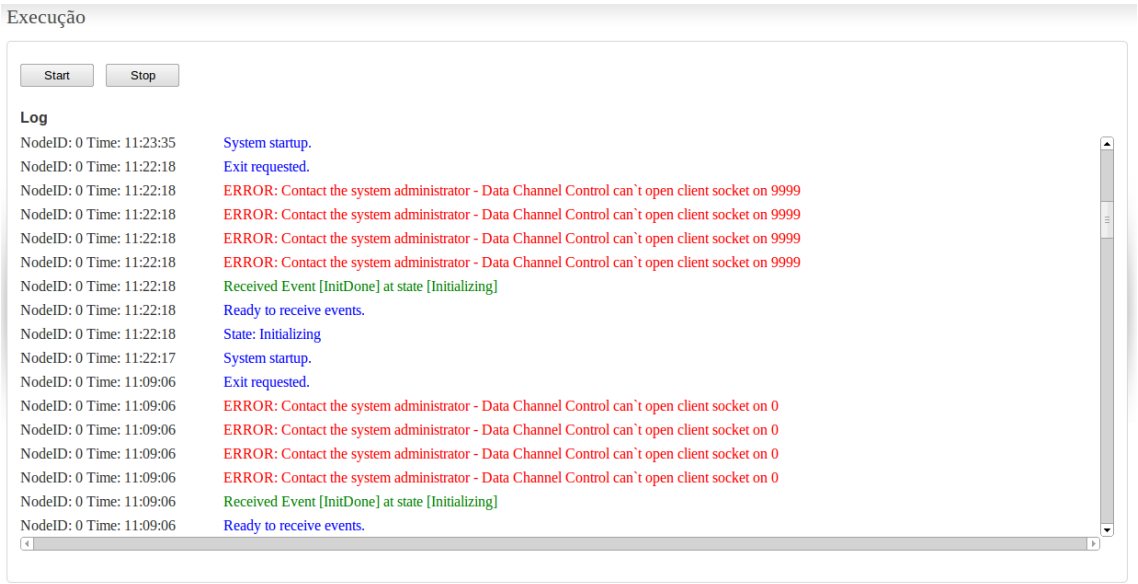

Figura 11. Tela de Monitoração

identificador do nó, o tempo em que ocorreu o log e a descrição do log em si. A cor azul foi definida para os tipos de log do sistema, a cor verde para os logs do teste e a vermelha para erros.

Principais classes

- Start
  - Servlet que envia comando para o controle iniciar o teste
  - Parâmetros
    - Request - entrada — Possui o identificador do teste
- Stop
  - Servlet que envia comando para o controle parar o teste
  - Parâmetros
    - Request - entrada — Possui o identificador do teste
- LerLog
  - Servlet que faz a leitura do log do teste
  - Parâmetros
    - Request - entrada — Possui o identificador do teste

1.2.5. Descrição do sistema – Controle Testbed

A seguir detalhamos os módulos do Controle Testbed: Motor de Execução, Interface de Controle e Interface de dados.

Motor de Execução O Motor de Execução está implementado como uma máquina de estados. Os eventos do sistema são centralizados por uma classe que utiliza o padrão de projeto *Singleton* para implementar uma fila de eventos. As classes auxiliares, quando

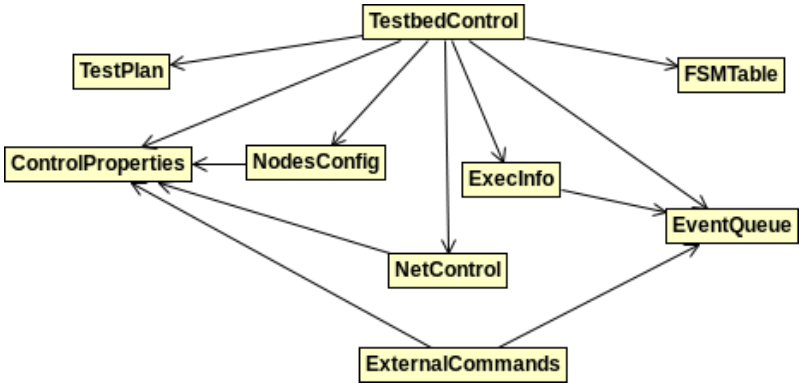

Figura 12. Diagrama de classes do Motor de Execução

necessário, geram eventos para a máquina de estados processar e disparar novas ações. Na Figura 12 apresentamos o diagrama simplificado das classes desse módulo.

Podemos descrever a execução do controle dos testes em três grupos de classes: Controle Geral, Acesso aos dados e Controle Remoto.

Controle geral Composto pelas classes *TestbedControl*, *FSMTable* e *EventQueue*. Responsável pelos passos de execução dos testes e também pela reação aos eventos do sistema. As respostas aos eventos dependem da informação da tabela de transições.

Na Figura 13 apresentamos o diagrama da máquina de estados implementado na classe *FSMTable*. Os estados têm os seguintes significados:

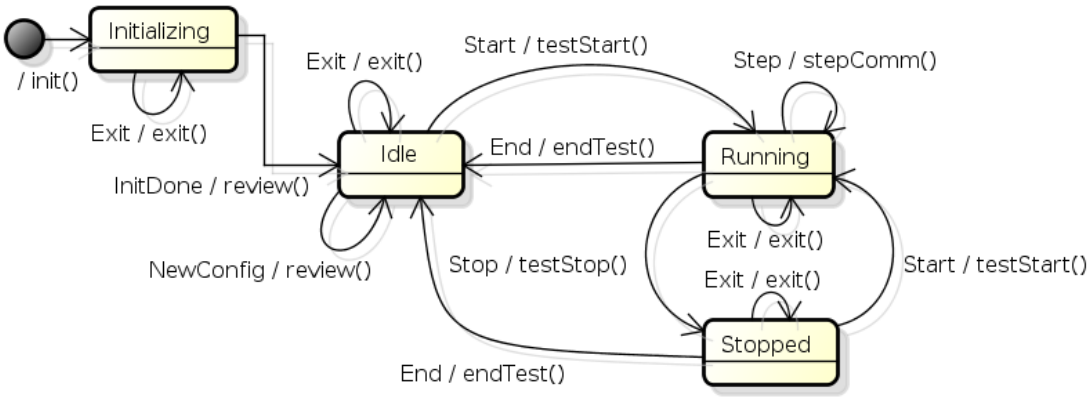

Figura 13. FSM do módulo de Controle de Execução

- *Initializing* — Estado da primeira inicialização do sistema de controle.
- *Idle* — O motor de execução está "desocupado" aguardando um evento de início de teste.
- *Running* — O motor está executando um teste programado.
- *Stopped* — O usuário solicitou a parada do seu teste via interface do portal.

As ações têm os seguintes significados:

- *exit()* - Fecha todas conexões e finaliza o programa.
- *review()* - Busca do banco a informação do teste corrente ou do próximo teste a ser executado.
- *testStart()* - Executa o teste corrente a partir do início do script de execução do respectivo teste.
- *stepComm()* - Executa o próximo passo de teste do script.
- *testStop()* - Para a execução do script de teste.
- *endTest()* - Finaliza a execução do teste corrente.

Os eventos têm os seguintes significados:

- *Exit* — É gerado quando ocorre um erro grave na inicialização ou quando o processo em execução recebe um "signal" do sistema operacional para interromper a execução.
- *InitDone* — Libera o controle de execução para novos testes.
- *NewConfig* — Evento gerado pelo portal quando um usuário salva uma nova configuração de teste. Serve para o motor verificar se o próximo teste foi alterado.
- *Start* — Evento gerado pelo temporizador de início de teste.
- *End* — Evento gerado pelo temporizador de final de teste.
- *Step* — Evento gerado pelo temporizador de passo de teste. Um passo de "ativação" ou "desativação" é executado imediatamente, enquanto que um passo "wait" é executado no período indicado pelo parâmetro.
- *Start* e *Stop* — São eventos gerados pelo usuário do teste via interface do portal.

Acesso aos dados As classes *TestPlan*, *NodesConfig*, *NetControl* e *ExecInfo* persistem as informações do banco de dados para configuração do teste corrente.

A classe *ControlProperties* persiste as informações de configuração do sistema que estão no arquivo */.TestbedControl.properties*. Nesse arquivo deve-se configurar as seguintes informações:

- Conexão do banco de dados.
- Diretório dos arquivos binários temporários e os comandos de conversão e carga dos binários.
- Número da porta telnet do SerialForward .
- Número da porta da interface remota do portal.
- Número inicial das portas TCP/IP de dados .
- Flag de controle para habilitar ou desabilitar o "trace" de execução.

Controle Remoto A classe *ExternalCommands* mantém o serviço TCP/IP para os comandos *NewConfig*, *Stop* e *Start* e gera os respectivos eventos para a FSM.

Interface de Controle da Rede A interface de controle da rede é baseada em três aquivos de comandos (shell script) para cada tipo de mote. Cada arquivo executa os comandos necessários para a ação e tipo de mote indicado. Esses arquivos ficam no diretório indicado dentro do arquivo de propriedades */.TestbedControl.properties*.

O sistema de controle é responsável pela passagem dos parâmetros "variáveis" para a execução das ações. Esses parâmetros estão armazenados na tabela *network* do banco de dados.

A seguir identificamos as três ações e os respectivos arquivos. O valor de 'x' depende do tipo de mote, sendo micaz=1, telosb=2:

- *convert_x.sh* — Prepara arquivo binário para ser carregado no mote e atualiza o ID lógico.
- *loadbin_x.sh* — Carrega o arquivo convertido no mote.
- *loadnoop_x.sh* — Carrega o arquivo binário padrão para desativar o mote.

Interface de dados da rede A interface de dados tem a função de redirecionar as portas de dados dos motes para portas TCP/IP locais da máquina do servidor, dessa forma o usuário pode ter acesso a esses canais a partir da sua máquina local.

A solução é baseada num programa python *Tbrelay.py* que funciona como um *relay* configurável entre portas TCP/IP. Esse programa consulta o arquivo de configuração *TBNet.cfg* para identificar quais canais devem ser ativados conforme a configuração da rede de sensores.

Adaptação para o dispositivo Silex SX3000g Os motes do tipo TelosB só possuem interface USB para comunicação com o mundo externo, desta forma se faz necessário adicionar um dispositivo conversor USB/Ethernet para viabilizar a conexão na rede do Testbed. Inicialmente optamos pela utilização do dispositivo SX3000g da Silex Technology. Uma restrição desse dispositivo é que o software do fabricante só funciona em ambiente Microsoft Windows. Como todo sistema do Testbed foi desenvolvido para ambiente Linux, foi necessário algumas adaptações para permirtir a conexão dos TelosB ao servidor de controle do testbed.

A nossa solução utiliza uma máquina virtual Windows instalada sobre o ambiente Linux. Nessa VM executamos o programa python *bsl_data_server.py* que é responsável pela comunicação e controle entre os scripts do Sistema de Controle no Linux e os programas/scripts de carga do executável no Windows. Esse programa python também é responsável pelo *relay* dos canais de dados no ambiente windows.

1.2.6. Configuração e Administração

Essa seção é direcionada ao administrador do sistema e informa os pontos de configuração necessários para o funcionamento do sistema. Algumas configurações são mais imutáveis, como as diretamente associadas à rede de sensores e tipos de usuários. Outras configurações são mais dinâmicas, como as associadas à configuração das agendas disponíveis ou criação de novos usuários. A seguir descrevemos esses pontos de configuração.

Rede física de sensores A rede física de sensores deve ser configurada em três pontos de configuração, sendo dois no banco de dados e um nos arquivos de comandos da interface de rede. Essas configurações devem estar coerentes entre si.

O primeiro ponto é a tabela *nodesTypes* no banco de dados. Deve-se configurar um registro para cada tipo de configuração de nó da rede. A configuração de instalação já vem preparada para dois tipos de nós, um nó com o mote MicaZ e a placa de sensoreamento MDA100 (nodetype=1) e o outro nó com o mote TelosB (nodetype=2).

O segundo ponto é a tabela *network*. Nessa tabela o administrador deve informar um registro para cada nó sensor da rede. O valor *nodeid* é o ID físico do nó na rede, o valor de *nodetype* deve ser preenchido com dados da tabela *nodesTypes*, o valor *ctlchannel* deve ser adequado para o comando de carga do mote e o valor *datachannel* deve ser adequado para o comando de canal de dados. Sugerimos utilizar como exemplo a configuração da instalação. Se for necessário criar novos tipos de motes, o valor de *ctlchannel* deve respeitar os parâmetros definidos para os novos arquivos de comandos de carga.

O terceiro ponto de configuração só será necessário atualizar se o sistema for utilizar novos tipos de motes. Os arquivos definidos no item 1.2.5 na página 21 devem ser utilizados como modelo. Deve-se criar o conjunto de três arquivos para cada tipo de mote. A sintaxe dos nomes de arquivos não pode ser alterada, respeitando o formato *action_x.sh*. Sendo *action* igual à "convert", "loadbin" ou "loadnoop" e *x* deve ser o respectivo valor de *nodetype*. Os comandos internos devem ser associados aos respectivos comandos para os novos tipos de motes. Esses comandos podem ser facilmente identificados quando executamos o comado *make <tipomote> install* numa instalação TinyOS 2.x.

Tipos de usuários A versão de instalação prevê dois tipos de usuário, "Professor" e "Aluno". Esses tipos são utilizados para discriminar o horário das agendas disponíveis. Essa discriminação é feita por nível de acesso, sendo o nível 100 para estudante e 500 para professor.

Se for necessário a criação de novos níveis, basta reconfigurar os níveis dos usuários existentes na tabela *users* e também reconfigurar os níveis das agendas disponíveis nas tabelas *slotstemplates* e *extraslots*.

Agenda padrão e eventual O sistema prevê a disponibilidade de slots baseado num padrão de horário semanal. Esse padrão deve ser configurado na tabela *slotstemplates*. Cada slot é identificado pelo valor de *slotoffset* que identifica um valor do tipo *YYYYMMDDHHMM*. No caso da tabela *slotstemplate* esse valor é um valor relativo para uma semana que começa na data e hora zero, isto é, o valor *000000000000* representa o primeiro slot do domingo da semana padrão. Os slots tem duração de 30 minutos, então o segundo slot de domingo terá o valor *000000000030*. O slot das 10h da terça-feira terá o valor *000000021000*. Também deve-se indicar o nível de acesso ao slot através do valor de *accesslevel*. Um slot pode ser desabilitado simplesmente fazendo o valor *enabled* igual a "false".

Caso necessite alterar a configuração para uma data específica, deve-se incluir novos registros na tabela *extraslots*. A configuração de *extraslots* sobrepõe automaticamente a configuração de *slotstemplate*.

Manutenção de usuários A manutenção dos dados dos usuários deverá ser feita pela tela "Manutenção de Usuários" conforme descrito no item 1.2.4.

A configuração inicial do sistema inclui um usuário administrador "admin". Somente usuários administradores possuem permissão para utilizar a interface de criação, atualização e remoção de usuários. Deve-se criar pelo menos um usuário comum para que o sistema possa ser utilizado.

1.2.7. Instalação

Nessa seção apresentamos um resumo da instalação do sistema, incluindo todos softwares necessários.

Principais Tecnologias Os seguintes softwares necessitam ser instalados:

- Sistemas Operacionais:
  - Ubuntu 12.04 LTS 32 bits
  - Windows 7 em Máquina Virtual
- Softwares:
  - java 1.7.0_45
  - VirtualBox 4.3.2
  - PostgreSQL 9.1.10
  - Pgadmin 1.14.0
  - Apache Tomcat 8
  - Python 2.7.7
  - Pyserial
  - Uisp Tools
  - Cygwin

Procedimento de instalação O kit disponibilizado para instalação do sistema possui os seguintes arquivos:

PortalTB.war - Pacote de instalação do Sistema Portal. Contém toda implementação do portal. Possui as páginas e servlets utilizados para o sistema.
TBControl_cygwin - Pacote de instalação do Sistema Controle na plataforma Windows
TBControl_linux - Pacote de instalação do Sistema Controle na plataforma Linux. Contém os módulos responsáveis pelo controle e execução dos testes.
BDPortal_backup.tar - Backup do banco de dados do testbed

O processo de instalação deve ser realizado na seguinte ordem:

1. Instalar o Sistema Operacional (Ubuntu 12.04 LTS 32 bits).
2. Instalar o Java (Java 1.7.0_45).
3. Instalar o banco de dados (PostgreSQL 9.1.10 ou superior).
4. Instalar o administrador do banco (PgAdmin versão 1.14.0 ou superior). Essa ferramenta serve de apoio ao desenvolvimento e manutenção. Sua instalação é recomendada, mas não é obrigatória.
5. Instalar Apache Tomcat (Apache Tomcat 7 ou superior).

6. Instalar Uisp Tools.
7. Instalar VirtualBox.
8. Instalar Windows 7 no VirtualBox.
9. Instalar Python e Pyserial.
10. Instalar Cygwin.

O roteiro completo de instalação será disponibilizado como um documento a parte.

1.3. Requisitos de hardware e software

Nesta seção apresentamos os requisitos de hardware e software para a instalação e os testes do protótipo.

1.3.1. Projeto físico do testbed de sensores e recursos de hardware necessários

A Figura 14 apresenta uma visão da arquitetura física do testbed.

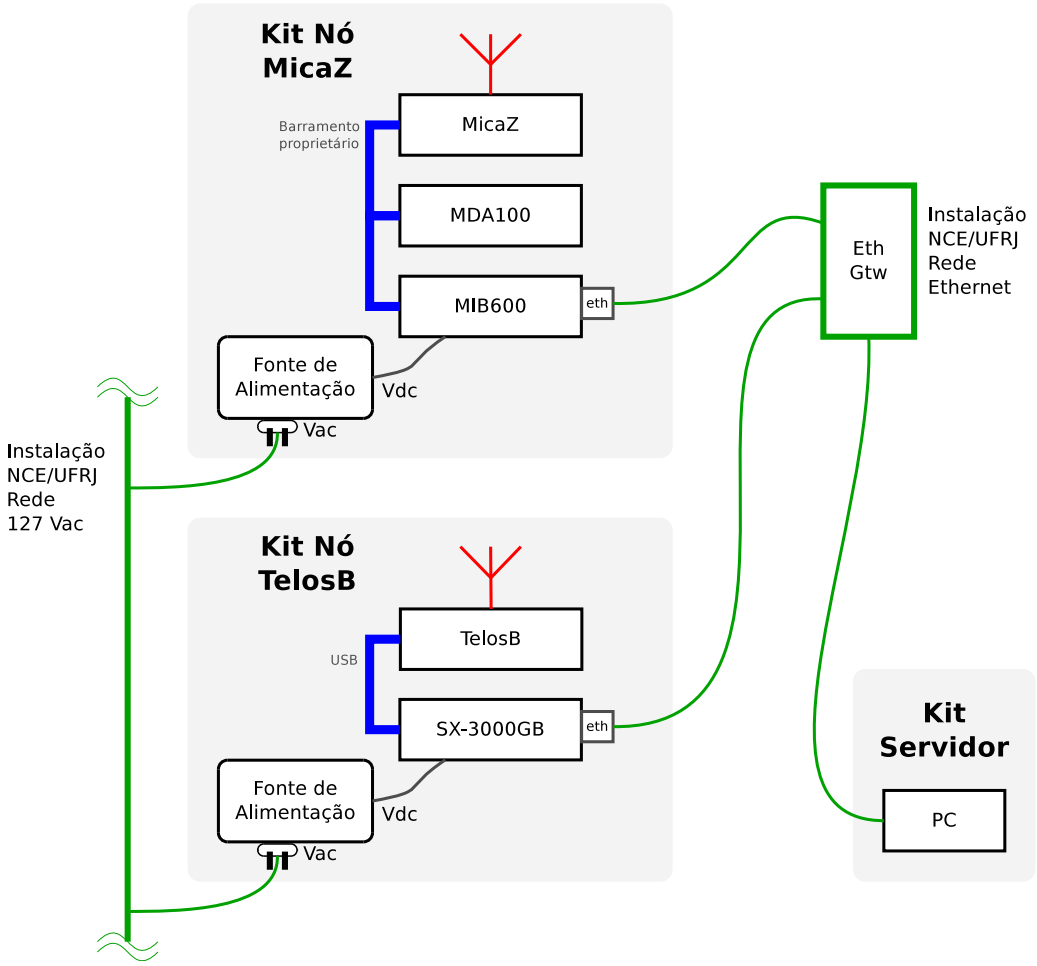

Figura 14. Projeto físico do testbed de sensores.

No nosso protótipo de testes temos dois tipos de Kit Nó, um com o mote **MicaZ** e o outro com o mote **TelosB**. A seguir especificamos cada um dos componentes dos dois tipos de Kit Nó.

O **Kit Nó MicaZ** é baseado em um mote padrão Berkeley, atualmente fabricado pela Memsic Corporation, com a seguinte configuração:

1. Módulo Processador + Rádio
    - Código fabricante - MPR2400CA
    - CPU - Atmel ATmega128L
    - ROM - 128K bytes
    - RAM - 4K bytes
    - Data - Flash memory 512K bytes
    - Rádio - Chipcon CC2420 - 2.4 GHz IEEE 802.15.4
    - Interface Controle - Conector Proprietário 51-pinos
2. Módulo Sensor
    - Código fabricante - MDA100CA
    - Sensor de temperatura
    - Sensor de luminosidade
    - Área para protótipo
    - Permite a conexão simultânea de mais 2 módulos com utilizando Conector Proprietário 51-pinos.
3. Módulo Interface de Rede
    - Código fabricante - MIB600
    - Conector RJ-45 100 Base
    - Interface IEEE 802.3 / IEEE 802.3af POE
    - Interface Controle - Conector Proprietário 51-pinos
    - Protocolos ARP, UDP/IP, TCP/IP, Telnet, DHCP, BOOTP, TFTP, Auto IP, and HTTP
4. Módulo de Alimentação
    - Disponibilizado em conjunto com o Módulo Interface de rede MIB600.
    - Compatível com o conjunto MicaZ + MDA100

Na Figura 15 apresentamos as imagens dos três módulos do Kit Nó MicaZ.

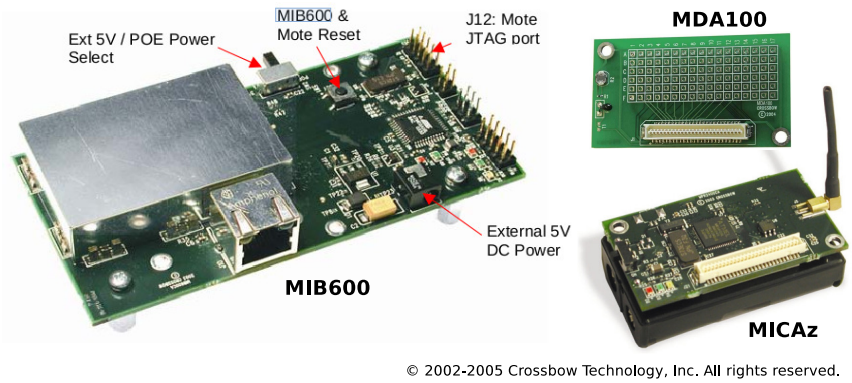

Figura 15. Componentes do Kit Nó MicaZ.

O **Kit Nó TelosB** é formado por componentes de dois fabricantes: o mote TelosB, que também é baseado num mote padrão Berkeley, atualmente fabricado pela Memsic Corporation;e o adaptador SX-3000GB fabricado pela Silex Technology. O kit tem a seguinte configuração:

1. Módulo Processador + Rádio + Sensor + Interface Controle
    - Código fabricante - TPR2420CA
    - CPU - TI MSP430
    - ROM - 48K bytes
    - RAM - 10K bytes
    - Data - Flash memory 1024K bytes
    - Data - EEPROM 16K bytes
    - Rádio - Chipcon CC2420 - 2.4 GHz IEEE 802.15.4
    - Interface Controle - Padrão USB
    - Sensor de temperatura
    - Sensor de luminosidade
    - Sensor de umidade
2. Módulo Interface rede
    - Código fabricante - SX-3000GB
    - Conector RJ-45 10 / 100 / 1000 Base
    - Interface IEEE 802.3
    - Interface USB 2.0 Hi-Speed (A-Type)
    - Protocolo TCP/IP,
3. Módulo de alimentação
    - Disponibilizado em conjunto com o Módulo Interface de rede SX-3000GB.

Na Figura 16 apresentamos as imagens dos dois módulos do Kit Nó TelosB.

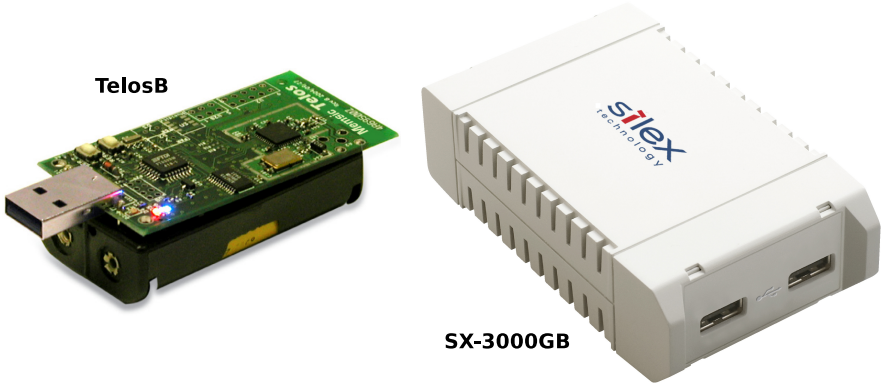

Figura 16. Componentes do Kit Nó TelosB.

Os módulos que compõem cada tipo de kit foram encapsulados em uma caixa plástica.

Distribuição física dos nós Atualmente, o nosso protótipo contém 7 kits Micaz e 5 kits Telosb dispostos em um laboratório do NCE/UFRJ.

2. Resultados dos testes

Como previsto em [Rossetto et al. 2014], realizamos os testes para validação do protótipo em três etapas. Na primeira etapa de avaliação, um conjunto de aplicações básicas de redes de sensores sem fio, foram submetidas para execução no testbed.

Na segunda etapa de avaliação, demos ênfase a aplicações que requerem integração da aplicação executando na rede de sensores com a Internet. Essas aplicações usaram a pilha de protocolos IPv6 adaptada para redes LoWPAN.

Na terceira etapa de avaliação, o testbed foi experimentado por desenvolvedores de aplicações para redes de sensores externos ao projeto (alunos de graduação e pós-graduação da PUC-Rio). Nessa etapa, avaliamos se o testbed é capaz de atender aos requisitos dos testes e avaliações projetados pelo desenvolvedor da aplicação. Todo o acesso ao testbed, incluindo a configuração dos experimentos, gerência da distribuição de tarefas entre os nós da rede, monitoramento e recebimento dos dados coletados foi realizado via portal Web.

Para cada etapa de teste relatada a seguir, reproduzimos os mesmos tópicos/aspectos do testbed que foram propostos para avaliação no RT-2. Os tópicos que não puderam ser avaliados por completo são comentados.

2.1. Aplicações básicas de redes de sensores sem fio

Na primeira etapa de testes, o objetivo foi avaliar de forma geral todas as funcionalidades do portal Web para acesso remoto ao testbed e recebimento dos logs de execução.

Um conjunto de aplicações básicas para redes de sensores sem fio já providas pelo framework TinyOS foram selecionadas e executadas no testbed. Havíamos previsto no RT-2 a inclusão também de testes de aplicações básicas desenvolvidas no Contiki, decidimos, entretanto, deixar os testes com aplicações Contiki apenas para a segunda etapa.

Avaliação Os sequintes aspectos foram avaliados nessa etapa de testes:

- **Facilidade de uso da interface do portal**. Verificou-se certa dificuldade para o usuário não familiarizado entender a sequência de passos que devem ser realizados para que um experimento seja configurado e executado. Mudanças na interface do portal já estão sendo implementadas para resolver essa demanda.
- **Funcionalidades providas pelo portal (controle de acesso, agendamento, configuração de experimentos, e execução)**. Os comandos da interface do portal responderam como esperado, mas a cada ação de um botão, por exemplo, a página é recarregada, assim, algumas informações que são interessantes para o usuário saem da tela. Isso não acontece na última tela (Plano de Execução), mas acontece nas outras telas (por exemplo, Topologia). Na tela de Topologia, nas opções "Alterar nome de arquivos" e "Deletar arquivos não utilizados" não apareceram todos os arquivos que foram carregados, apenas o último.
  O controle de acesso é importante para que usuários não interfiram nos experimentos uns dos outros. Entretanto, na versão atual do protótipo qualquer usuário que souber o endereço IP e porta dos nós que estão executando o experimento poderá

ter acesso ao que está sendo enviado pela serial do dispositivo. Contactamos especialistas da área de segurança da informação para solicitar ajuda na solução dessa demanda.

Por fim, avaliamos que foi uma boa ideia oferecer a opção de criar um script para ativar/desativar a execução do nós, e definir o intervalo de tempo de espera entre cada ação. Ajudou a controlar a ocorrência de determinados eventos e observar como as aplicações reagem a eles.

- **Operação dos nós do testbed (corretude, tempo para carga das aplicações e coleta dos resultados, interferências do meio)**: Os nós do testbed funcionaram corretamente, o tempo de carga das aplicações e coleta dos resultados ficou dentro dos limites esperados e não detectou-se interferência do meio.
- **Interfaceamento do testbed com a máquina onde executa a aplicação do usuário, via interface serial**: Atualmente há duas opções para a coleta de dados, via interface serial e por um LOG de execução online. Quanto ao LOG de execução, as informações apresentadas ainda estão um pouco confusas, o que dificulta a depuração da execução de experimentos mais complexos. Talvez imprimir o conteúdo dos dados que estão passando pela serial, como é feito hoje, não seja necessário. Apenas informar o tamanho desses dados já seria suficiente, visto que é a aplicação do usuário que deve se preocupar em entender esses dados.

Um levantamento geral de sugestões de melhorias na interface do portal foi realizado e parte dessas sugestões estão sendo implementadas para a versão final do portal.

2.2. Aplicações para demonstração de uso da pilha IPv6

Na segunda etapa do testes, o objetivo foi demonstrar e avaliar a execução de aplicações para redes de sensores sem fio que fazem uso da pilha de protocolos IPv6. Duas aplicações alvo, baseadas em comunicação via IPv6, foram desenvolvidas para essa etapa de testes.

A aplicação *SmartHome* segue um modelo Request/Response. Uma aplicação Java executando remotamente na máquina do usuário interage com os motes através de requisições, simulando a interação com elementos de uma casa inteligente. Na aplicação *WaterMonitor*, os motes do testbed coletam dados de temperatura, umidade e luminosidade e enviam para um programa Java que executa na máquina do usuário. Essa aplicação segue o modelo de comunicação *Push*. O projeto detalhado dessas aplicações foi descrito em [Rossetto et al. 2014] (RT2).

Nesta etapa, utilizamos apenas os microcontroladores TelosB pois são os únicos motes disponíveis no protótipo de avaliação do testbed que comportam as aplicações IPv6 (restrição de memória).

Avaliação Os sequintes aspectos foram avaliados nessa etapa de testes:

- **Interfaceamento do testbed com a máquina onde executa a aplicação do usuário, via protocolos IP**. A forma de interação com a interface serial para montar a conexão de rede IPv6 depende da plataforma em que se está programando. O TinyOS utiliza o protocolo PPP, enquanto o Contiki utiliza o SLIP. Essa conexão é necessária para que o Contiki e o TinyOS possam enviar e receber os pacotes da RSSF. Os dois sistemas oferecem uma aplicação que realiza o roteamento de pacotes que vem de uma rede externa para a rede de sensores sem fio e vice-versa.

Todos os testes que foram realizados utilizaram essas ferramentas para a comunicação entre aplicações executando no computador e aplicações executando nos dispositivos da rede de sensores sem fio. Apenas o TinyOS apresentou algumas falhas na conexão entre a máquina remota do usuário com o mote estação base do testbed. No entanto, de forma geral, uma nova tentativa resolvia o problema.
As aplicações de teste funcionaram como esperado. Isso mostra que a atual arquitetura do testbed pode ser empregada para o teste de aplicações utilizando IPv6, tanto no Contiki como no TinyOS.

- **Tempo de resposta das aplicações (comunicação entre estação base e máquina do usuário)**. O tempo de carga dos códigos Contiki nos microcontroladores TelosB consumiu de 70 a 90 segundos em média. Enquanto que a carga no MicaZ consumiu um tempo comparativamente menor. Não chegamos a realizar medidas de tempo de comunicação entre o nó estação base e a máquina do usuário durante a execução da aplicação.
- **Desempenho dos protocolos IPv6 dentro da rede de sensores**. O tempo de comunicação entre os microcontroladores para as ações de descoberta de vizinhos e criação da árvore de rotamento RPL variou entre 60 e 120 segundos, dependendo da quantidade de microcontroladores envolvidos no experimento. É necessário avaliar ainda qual parcela desse tempo é requerida especificamente pelo algoritmo implementado pelo Contiki e qual parcela é devida a carga remoto de código no testbed. A montagem da árvore de roteamente RPL no TinyOS consumiu um tempo menor.
  Os testes realizados no testbed responderam de mesma forma que os testes realizados no simulador COOJA. As diferenças foram o tempo de resposta e a construção da árvore de roteamento RPL. Quanto ao tempo de resposta, levou-se mais tempo para obter respostas de aplicações que executavam no testbed do que aplicações que executavam no simulador. Mas isso já era esperado, em testes com o simulador tanto a aplicação do usuário quanto a rede de sensores sem fio estavam na mesma máquina local. Logo, não havia atraso de roteamento nem a distância física para aumentar o tempo de resposta. Já nos testes como o testbed, a distância física e outras características de uma rede convencional se destacaram, o que foi bom para avaliar o comportamento das aplicações em um ambiente real e coletar dados reais, que são mais interessantes do que dados gerados por um simulador.
  Outras medições, incluindo: taxa de entrega dos pacotes, RTT, taxa de perda de pacotes, fragmentação de pacotes, auto-configuração das rotas, atribuição de endereços IPv6 aos nós sensores, escalabilidade e tolerância a falhas serão realizadas posteriormente pois dependem principalmente dos algoritmos implementados pelo TinyOS e Contiki e não do acesso ao testbed em si.

2.3. Acesso ao testbed por outros grupos de pesquisa

Na terceira etapa de testes, o objetivo foi realizar uma avaliação geral do testbed por usuários externos ao projeto. Para isso, o testbed foi usado como ferramenta de experimentação pelos alunos a disciplina Sistemas Distribuídos do programa de pós-graduação em Informática da PUC-Rio. No semestre acadêmico de 2014.1, a disciplina utilizou o testbed na elaboração de dois trabalhos (entregues, respectivamente, no início e final do mês de junho). Os alunos receberam uma máquina virtual preparada com o ambiente de execução e simulação de programas Terra (ambiente em desenvolvimento na PUC-Rio

para facilitar o desenvolvimento de aplicações para redes de sensores sem fio), e foram orientados a desenvolver um programa com difusão e coleta de informação na rede de sensores sem fio. No segundo trabalho, esse programa passou a interagir com outro, em execução na Internet tradicional, formando uma aplicação híbrida. A turma tinha dez alunos, distribuídos entre o final da graduação e o início do Doutorado. Todos executaram as tarefas solicitadas e vários deles relataram as principais impressões e dificuldades na entrega dos trabalhos.

A seguir, resumimos essas impressões, combinado-as com as observações da professora, e classificando-as de acordo com os aspectos que havíamos enumerado no RT2.

Avaliação Os sequintes aspectos foram avaliados nessa etapa de testes:

- **Facilidade de uso da interface do portal**: De maneira geral, os alunos não tiveram dificuldades com o uso da interface. Um único aluno relatou dificuldades detectar se os nós estariam prontos a carregar um novo programa.
- **Funcionalidades providas pelo portal (controle de acesso, agendamento, configuração de experimentos, e execução)**: Vários alunos elogiaram explicitamente a possibilidade de execução de seus programas em uma plataforma real, observando que isso tornou o trabalho mais motivador ou instigante. Por outro lado, uma dificuldade também apontada por vários diz respeito à depuração de programas. A depuração de programas em uma RSSF é tarefa notoriamente difícil e no testbed perde-se a pequena ajuda de observar os leds. Alguns dos estudantes propuseram soluções que poderiam de fato facilitar essa tarefa.
  Uma outra observação feita por mais de um aluno foi em relação à topologia do tesbed, na qual no momento todos os nós "enxergam" todos os demais. Os alunos sentiram a necessidade de testar seu trabalho com outras topologias, sendo que alguns deles chegaram a simular outras situações dentro do software. Dada a escassez de recursos de processamento, essa solução é muito onerosa para ser considerada prática.
- **Disponibilidade do portal e do testbed**: Nenhum aluno relatou problemas com isso. Em uma das entregas houve um dia de desconexão do portal por problemas de rede no local onde fica instalado, fazendo com que a entrega do trabalho fosse adiada por conta da professora, antes mesmo de qualquer reclamação por parte dos alunos.
- **Operação dos nós do testbed (tempo para carga das aplicações e coleta dos resultados)**: Os nós do testbed funcionaram safisfatoriamente.
- **Interfaceamento do testbed com a máquina onde executa a aplicação do usuário, via interface serial e protocolos IP**: Houve alguns elogios explícitos ao SForward (método de transferência da interface de comunicação serial dos motes para portas IP). No segundo trabalho, os alunos interfacearam o programa RSSF com uma aplicação Lua rodando em suas máquinas. Para isso, utilizaram com sucesso a biblioteca Lua `tossam` (`www.inf.ufg.br/~brunoos/tossam/`) que permite enviar e receber mensagens TinyOS na porta serial.
- **Solução de controle de acesso federado ao testbed**: Como ainda estamos desenvolvendo a solução de acesso federado ao testbed, não foi possível avaliar esse item.

Nossa planejamento inicial para essa etapa de testes era incluir também usuários de outros grupos de pesquisa do Brasil, o que não foi possível realizar ainda. Completaremos essa etapa de teste ao longo dos meses de agosto e setembro. Serão elaborados questionários para que os usuários externos avaliem o testbed.

3. Avaliação

De forma geral, avaliamos que o testbed desenvolvido (portal de acesso e gerência da rede de sensores) atendeu às expectativas que tínhamos no início do projeto. Além disso foi possível confirmar a viabilidade de experimentar aplicações híbridas no testbed, as quais executam em parte nos nós sensores e em parte em outros computadores ligados à Internet. Por fim, o protótipo também permitiu avaliar o uso de protocolos IPv6 no desenvolvimento de aplicações para os nós sensores (plataformas TelosB). Vários aspectos do protótipo foram avaliados e a necessidade de melhorias futuras em todos esses aspectos foram levantadas.

Acreditamos que o protótipo tem potencial para se tornar um serviço da RNP como ambiente para experimentação de aplicações distribuídas híbridas que requerem a integração entre nós sensores e outros dispositivos convencionais conectados à Internet. Esse ambiente de experimentação poderá servir tanto para atividades de pesquisa (por diferentes grupos de pesquisa no Brasil), permitindo a execução e reprodução de experimentos, quanto para atividades de ensino nas áreas convencionais de Redes de Computadores e Sistemas Distribuídos

A integração das redes de sensores com a Internet permite expandir as posibilidades de controle e acesso remoto aos chamados “espaços inteligentes” (monitorados por sensores) através do uso de protocolos de comunicação convencionais. Por outro lado, os sensores também podem ser considerados "coisas"(i.e., não são computadores convencionais) e por isso a capacidade de implementar/experimentar a pilha de protocolos IP nesses dispositivos constitui-se, de certa forma, em uma realização do conceito de Internet das Coisas. Dispor de um ambiente de experiementação físico nessa área é fundamental para a viabilização dos avanços tecnológicos.

Com relação ao formato de gerência desse serviço pela RNP, sugerimos inicialmente o apoio a um laboratório, com um espaço físico adequado, que mantivesse o testbed disponível e acessível por usuários remotos. Em paralelo, deveria-se estimular outras instituições a se integrarem como provedores de testbeds locais.

Outra direção seria integrar o testbed de sensores com outros testbeds para Internet do Futuro, tomando como base outros projetos em andamento, como é o caso do FIBRE.

Referências

Rossetto, S., Silvestre, B., Rodriguez, N., Branco, A., Kaplan, L., Lopes, I., Santana, D., da Silva Lima, V. B., de Freitas, R. G. M., and Aoki, D. T. (2014). GT-TeI: RT2 – proposta do protótipo. Technical report, RNP.

Werner-Allen, G., Swieskowski, P., and Welsh, M. (2005). Motelab: a wireless sensor network testbed. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN ’05, Piscataway, NJ, USA. IEEE Press.

WISEBED (2008). Design of the hardware infrastructure, architecture of the software infrastructure & design of library of algorithms. Technical report, Collaborative project, Small and medium-scale focused research project (STREP).