

Universidade Estadual do Centro-Oeste –
UNICENTRO Setor de Ciências Exatas e Tecnologia –
SEET Departamento de Ciência da Computação –
DECOMP
DISCIPLINA:2327– Engenharia de Software II – SÉRIE: 2

Prof. Dr. Marcos Antonio Quináia

Terceira etapa do
desenvolvimento de
software



Gabriel Lucas Fedacz,
João Felipe Pizzolotto
Bini e Juliano Carvalho.

Guarapuava, 20 de
Setembro de 2014

Introdução

Neste trabalho será apresentado o projeto de um sistema chamado SleepTree, que é um software que tem como ideia principal calcular as unidades de dormência de árvores frutíferas. O software recebe algumas temperaturas diárias e a partir destas é feita a interpolação para deduzir as temperaturas do resto do dia. Após encontradas as temperaturas o software utiliza fórmulas matemáticas para somar a quantidade de unidades de frio obtidas no dia.

Neste trabalho estão listados os requisitos do software, o cronograma de desenvolvimento, os casos de uso, diagramas de sequência e de classes, protótipos das janelas, ferramentas utilizadas no projeto, estudo e projeto da arquitetura de software e sobre a manutenibilidade e evolução, previsões das inspeções, banco de dados, casos de testes e estudo sobre confiança e proteção do software.



1. Levantamento de Requisitos do Software

Segue abaixo a tabela de requisitos seguindo modelo de Sommerville[1].

Requisitos do usuário	Requisitos do sistema
1. Contagem de unidades de frio seguindo os 3 modelos (Utah modificado, Carolina do Norte modificado, Padrão).	<p>1. Necessário inserir todos os dias a temperatura máxima, mínima e das 21 horas do dia para realizar a interpolação ou somente máxima e mínima destas para calcular a média das temperaturas de todos os outros horários do dia e poder calcular quantas unidades de frio foi obtida.</p> <p>1.2. Após obter as temperaturas das 24 horas do dia, utilizar os cálculos necessários para apresentar os resultados nos seguintes modelos: Utah Modificado, Carolina do Norte Modificado, Padrão.</p> <p>1.3. Armazenar todos os resultados em um banco de dados.</p>
2. Gerar relatórios diários, semanais, mensais, incluindo gráficos para melhor leitura.	<p>2. Gerar relatórios diários, semanais, mensais. Armazenando os dados em um banco de dados para futuras leituras dos dados.</p> <p>2.2. Gerar relatórios em forma de gráficos com os três modelos para poder comparar mais facilmente os métodos.</p>
3. Adicionar regiões, para poder calcular as unidades de frio de várias áreas.	3. O Programa deve oferecer a opção de criar uma região, ou escolher uma criada anteriormente para a inserção das temperaturas.
4. Possibilidade de importar tabelas de dados do SIMEPAR.	<p>4. Criar uma opção para importar tabelas do Excel para dentro do software e processar todos os dados para calcular as unidades de frio de um certo período de tempo.</p> <p>4.1. Gerar os relatórios e gráficos dos dados importados.</p>

1.3. Modelagem de Casos de Uso

Cada caso de uso representa uma tarefa discreta que envolve a interação externa com um sistema. Diagramas de casos de uso dão uma visão simples de uma interação. Logo, é necessário fornecer mais detalhes para entender o que está envolvido.

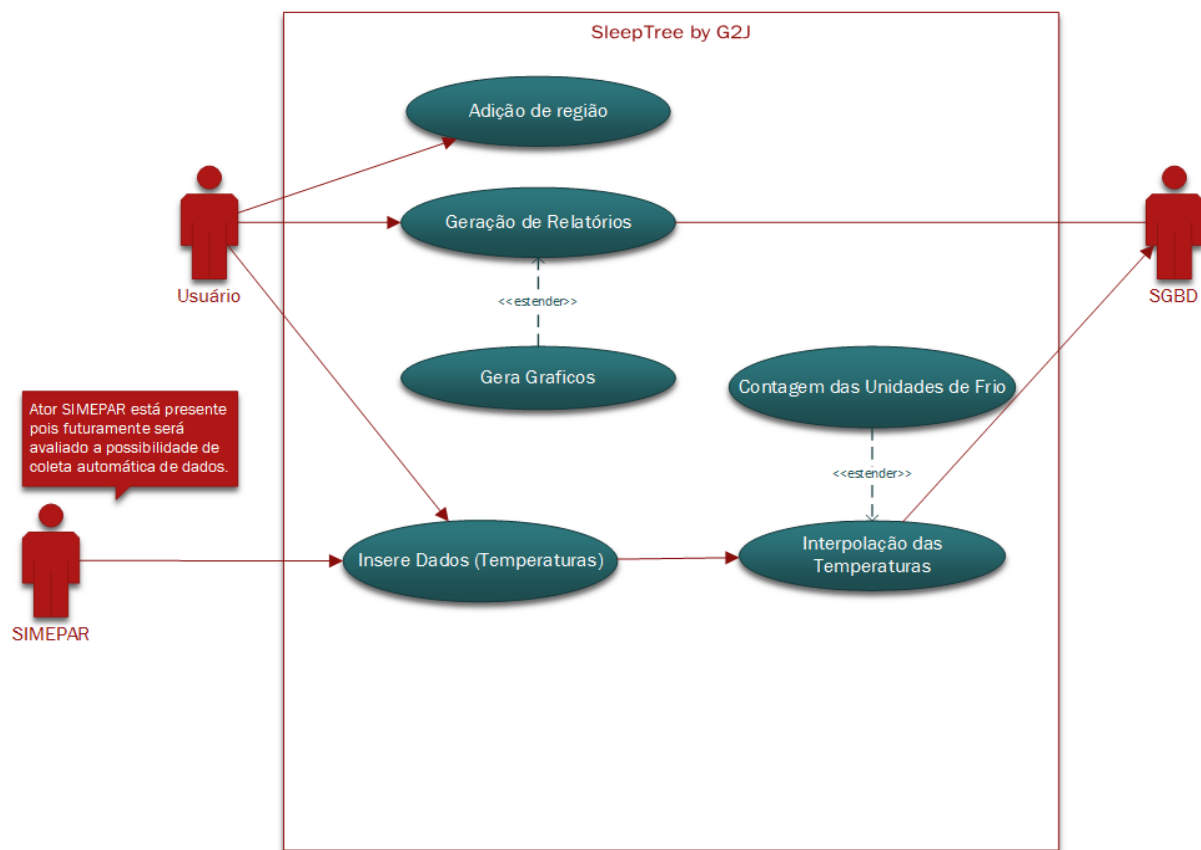


Figura 1 - Modelo de casos de uso

Nr	Caso de Uso	Quem Inicia a Ação	Descrição do Caso de Uso
1	Adição de Região	Usuário	Usuário cadastra uma região no SleepTree para poder começar um cálculo de Unidades de Frio.
2	Insere Dados	Usuário	Usuário insere dados na janela de temperaturas para o SleepTree fazer o cálculo de interpolação das temperaturas e fazer o cálculo das unidades de frio nos três modelos mencionados na elicitação de requisitos.
3	Geração de Relatórios	Usuário	Usuário seleciona a opção de gerar relatórios, assim sendo apresentado na tela os relatórios de uma ou mais áreas desejadas, oferecendo a opção de exportar os dados para o Excel ou para impressão.
4	Interpolação das Temperaturas	Insere Dados	Após o usuário inserir os dados no programa, será feita a interpolação das temperaturas inseridas e feito o cálculo de unidades de frio. Em seguida será exibido na tela os resultados e gravado no banco de dados todas as informações.

5*	Insere Dados	SIMEPAR	SleepTree irá pegar automaticamente as temperaturas fornecidas pelo sistema da SIMEPAR e efetuar os cálculos de interpolação e unidades de frio e gravar tudo no banco de dados.
----	--------------	---------	--

1.3.2. Modelagem diagrama de sequência

Os diagramas de sequência são usados, principalmente, para modelar as interações entre os atores e os objetos em um sistema e as interações entre os próprios objetos. Um diagrama de sequência mostra a sequência de interações que ocorrem durante um caso de uso em particular ou em uma instância de caso de uso [1].

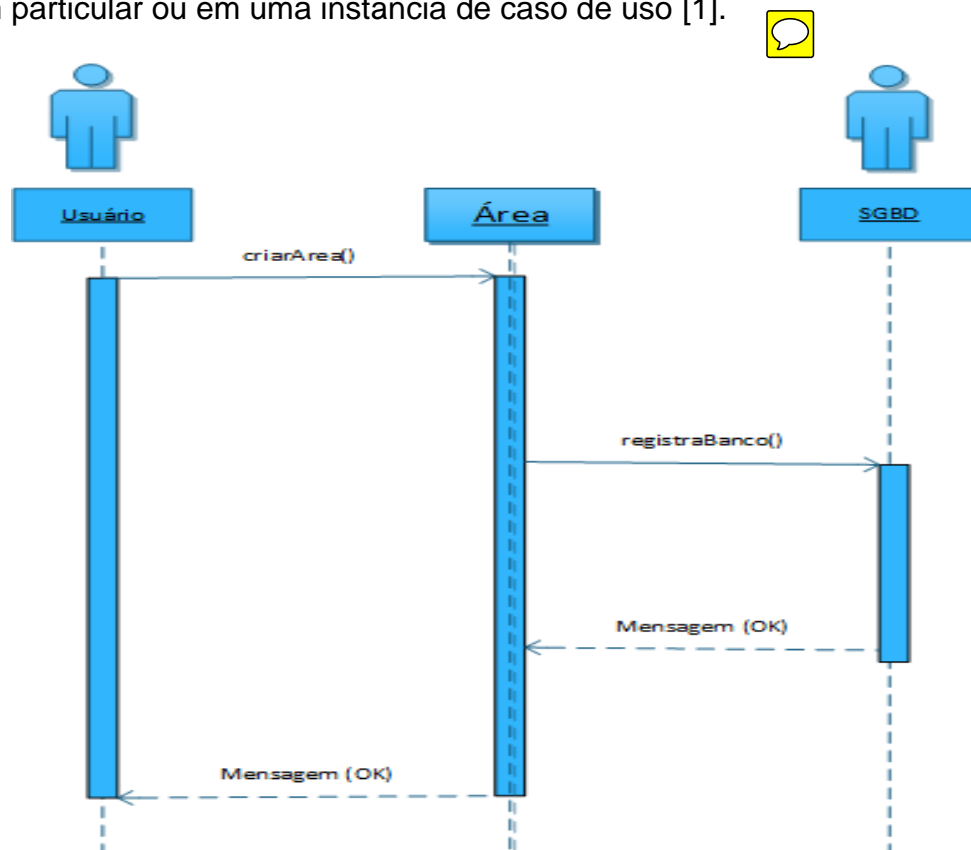


Figura 2 - Modelo de sequência de adição de região

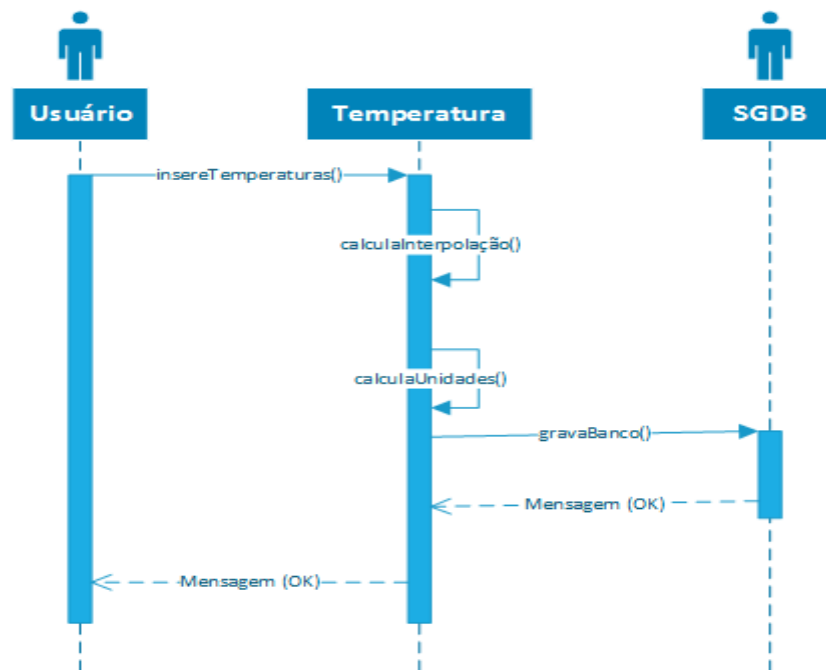


Figura 3 - Modelo de sequência de inserir temperaturas

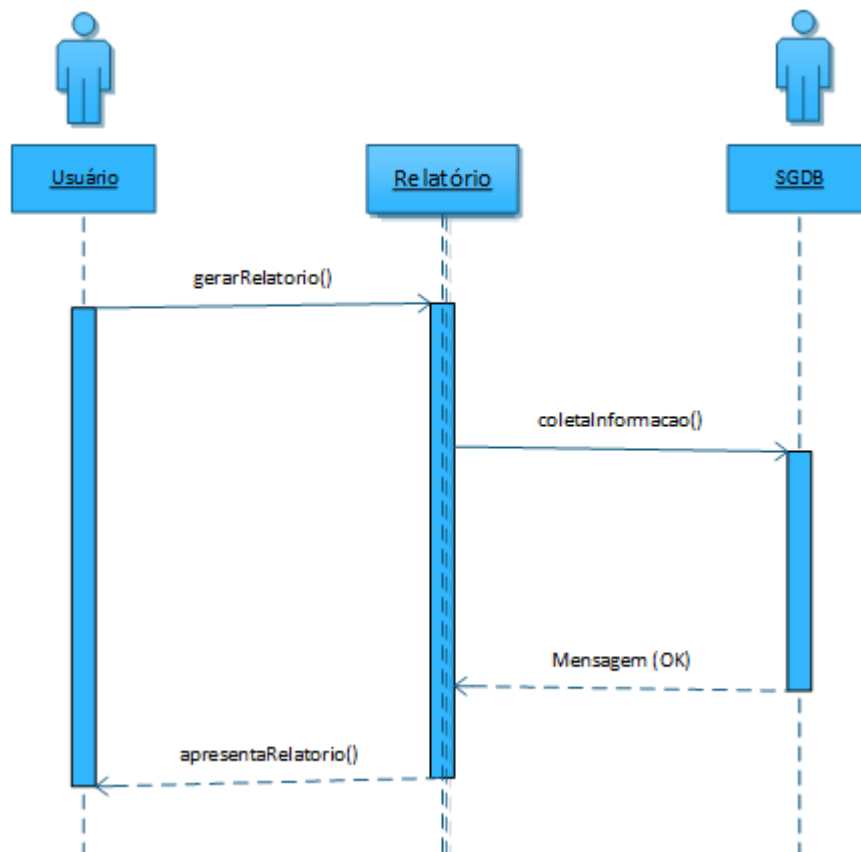


Figura 4 - Modelo de sequência para gerar relatórios

- **Modelo de sequência para coleta de dados do SIMEPAR**

No diagrama da figura 5 será estudado para ver se será possível implementar em nosso sistema, pois esta opção não depende somente de nós mas sim também da SIMEPAR,



pois teremos que ver se ela pode nos disponibilizar um banco de dados com as temperaturas oficiais do dia em tempo real.

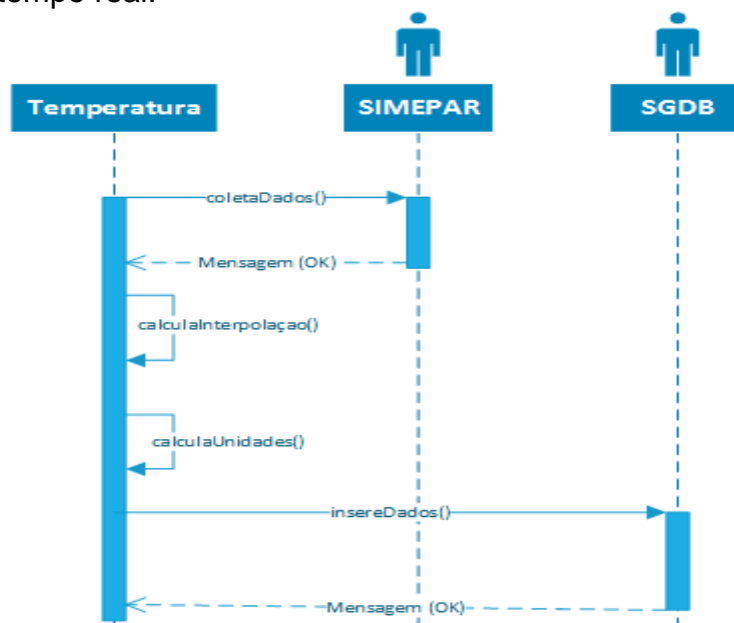


Figura 5 - Modelo de sequência para coleta de dados do SIMEPAR

1.3.3. Diagrama de Classes

Os diagramas de classe são usados no desenvolvimento de um modelo de sistema orientado a objetos para mostrar as classes de um sistema e as associações entre essas classes. Uma classe de objeto pode ser pensada como uma definição geral de um tipo de objeto do sistema.

O diagrama de classes da Figura 6 está apresentado os pacotes recolhidos da arquitetura MVC utilizado em nosso projeto que será utilizado para o desenvolvimento do software.

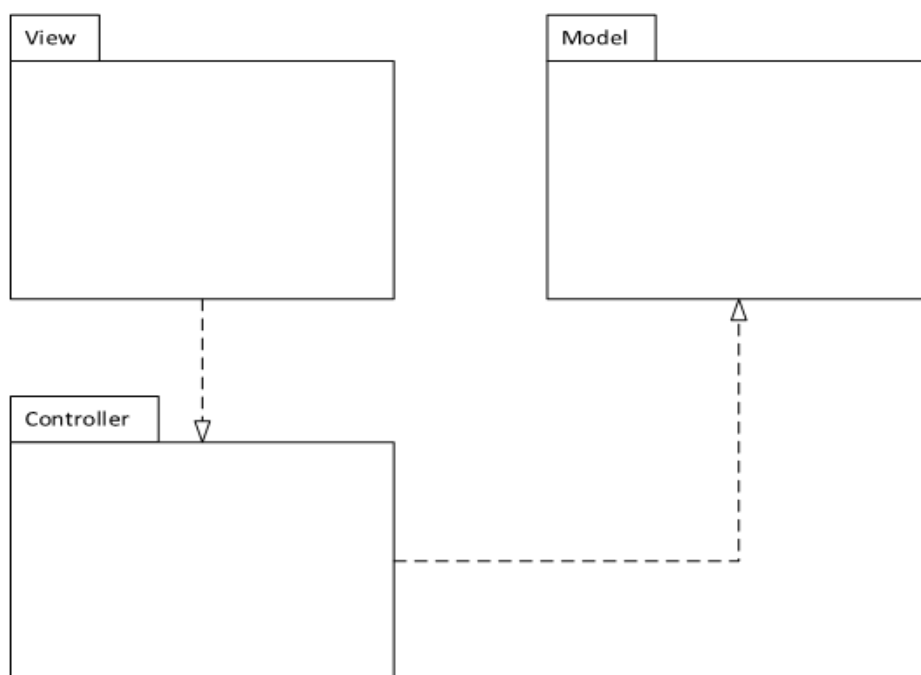


Figura 6 - Diagrama de classes reduzido da arquitetura MVC.

O diagrama apresentado no pacote View pode ser visto na Figura 7.

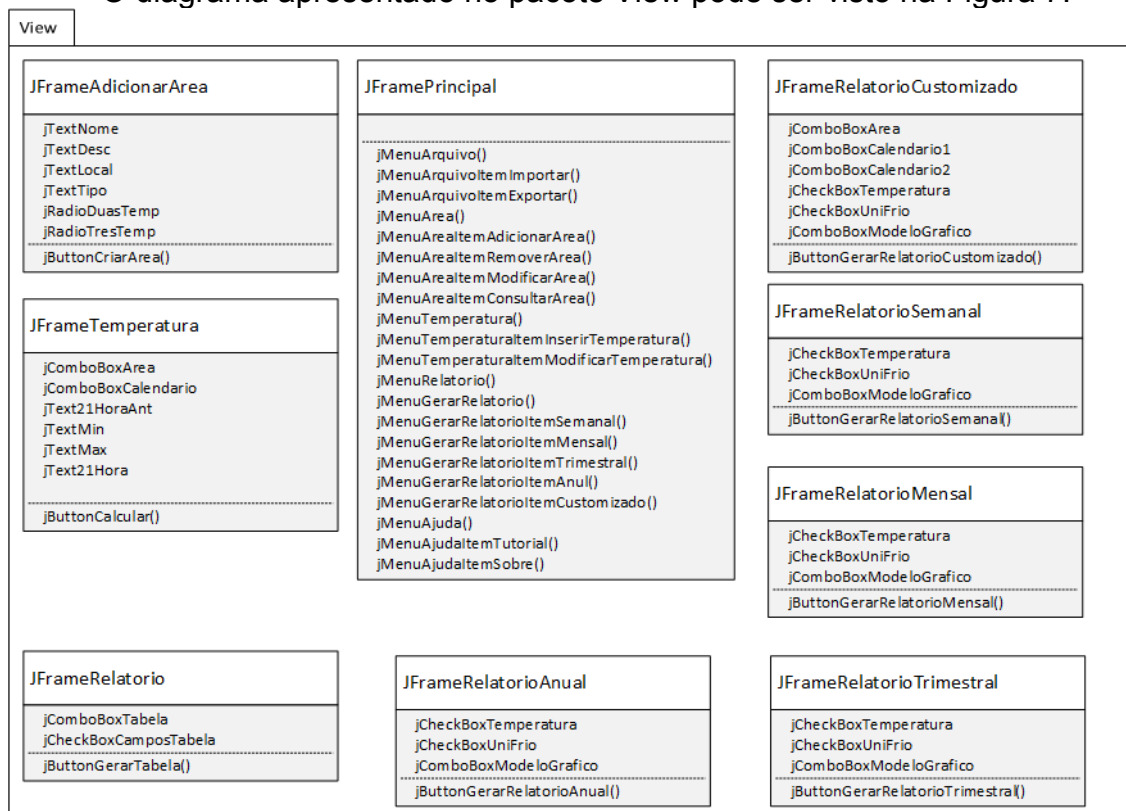


Figura 7 - Diagrama de classes do pacote View expandido.

8.

O diagrama apresentado no pacote Controller pode ser visto expandido na Figura

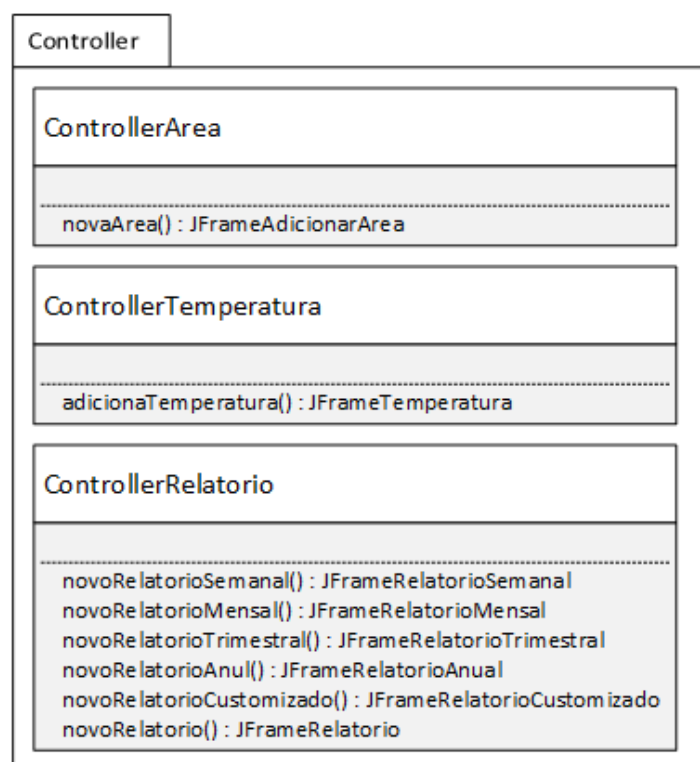


Figura 8 – Diagrama de classe do pacote Controller expandido.

O diagrama apresentando no pacote Model pode ser visto expandido na Figura 9.

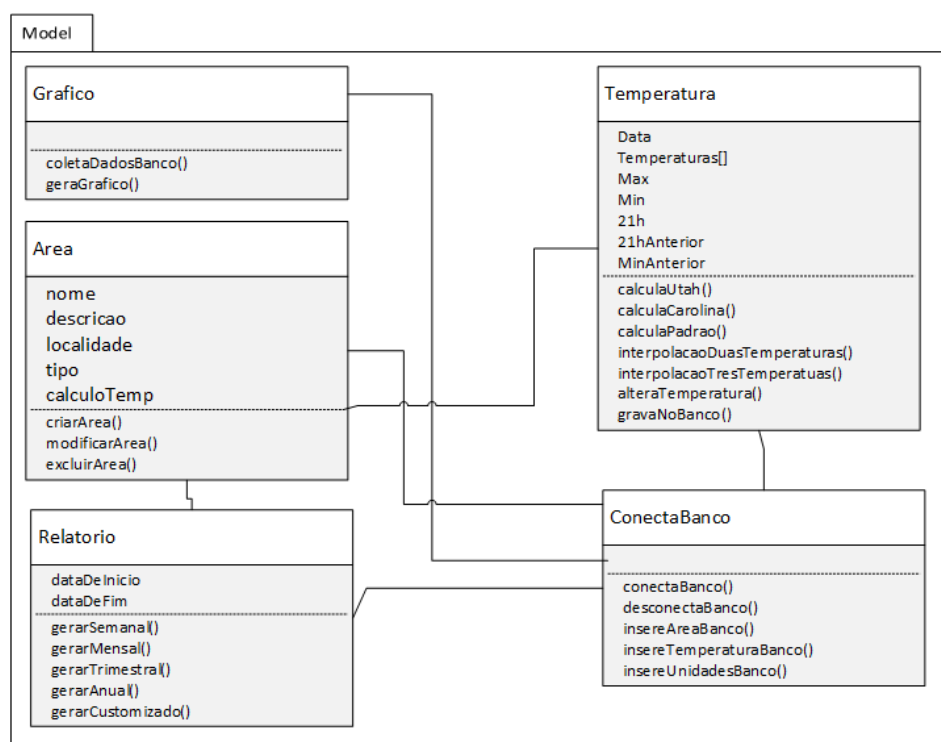


Figura 9 - Diagrama de classe do pacote Model expandido.

1.4 Protótipos das Janelas

A figura 10 mostra a janela principal do programa, com fácil acesso aos menus que são utilizados para criação de áreas, manipulação das temperaturas, geração de relatórios, importação e exportação de dados e ajuda ao usuário.

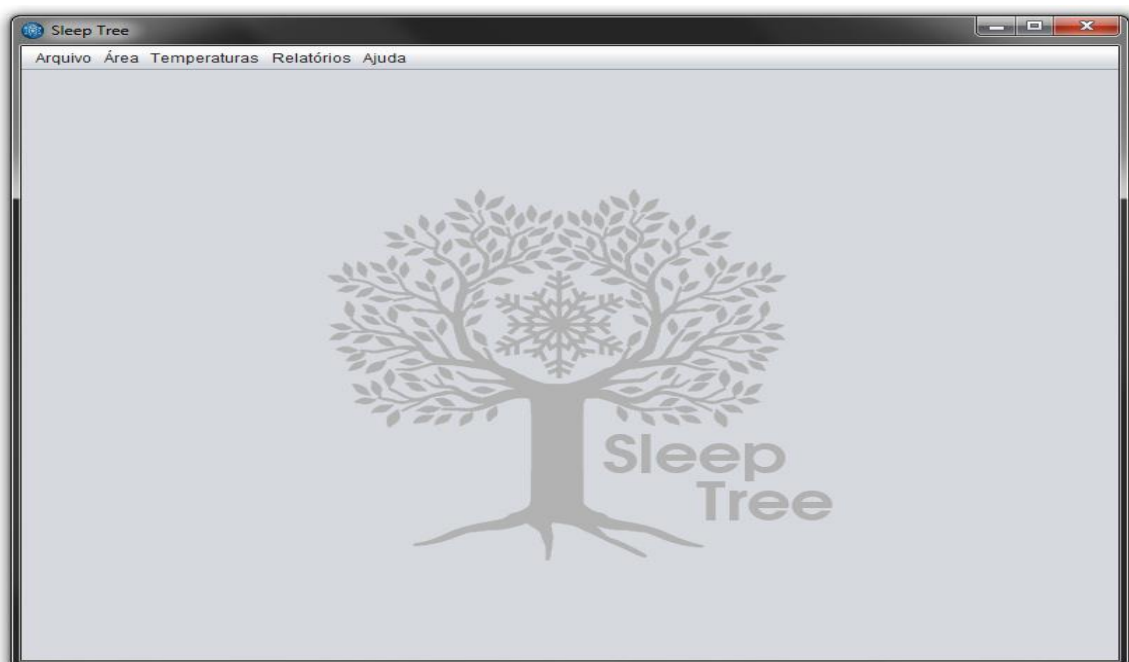


Figura 10 - Janela Principal

Na figura 11 é mostrada a janela de criação de áreas.

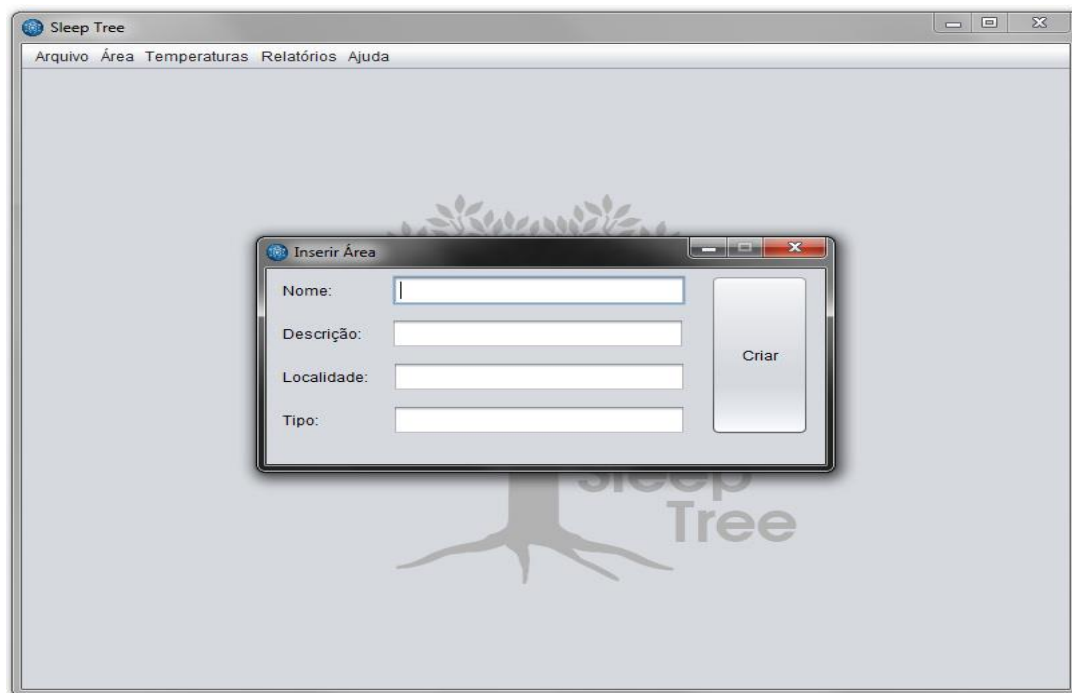


Figura 11 – Adição de região

A figura 12 mostra a inserção de temperaturas, o usuário deve selecionar uma área específica e então inserir as temperaturas.

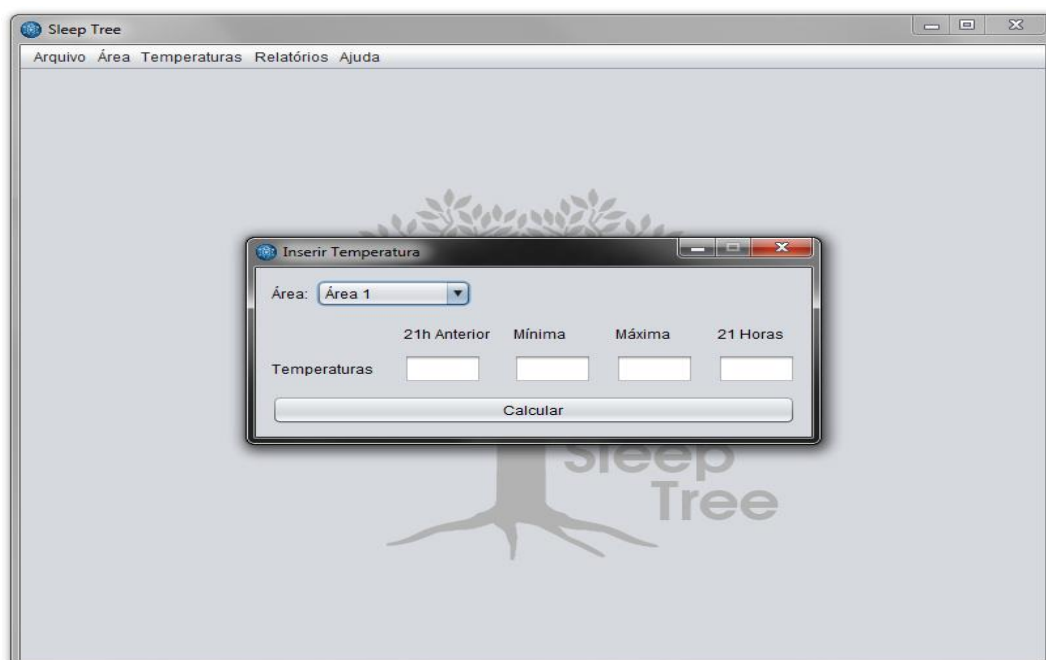


Figura 12 - Inserção de Temperaturas

2. Ferramentas para o projeto e desenvolvimento

O desenvolvimento de software é uma atividade complexa, envolvendo inúmeros fatores que são imprevisíveis e de difícil controle como mudanças de requisitos e prazos. São necessárias diversas ferramentas, que oferecem recursos para o desenvolvimento de software.

Abaixo são citados todos os softwares utilizados no desenvolvimento do projeto, como linguagens, IDE e ferramentas para auxílio e desenvolvimento do projeto.

- Desenvolvimento do software e geração de protótipos de janelas: Netbeans[2].
- Desenvolvimento dos diagramas ER de banco de dados: BrModelo[3] e Microsoft Visio[4].
- Criação de tabelas e campos do banco de dados: MySQL Workbench[5] e Postgree[6].
- Criação de cronograma: Microsoft Project Manager[7].
- Criação de documentos do software e requisitos: Microsoft Office Word[8].
- Criação de algoritmos de teste: CodeBlocks[9].
- Desenvolvimento do trabalho da equipe: Google Drive Docs[10] e Skype[11].

3. Cronograma de Execução do Projeto

O cronograma é um instrumento de planejamento e controle semelhante a um diagrama, em que são definidas e detalhadas minuciosamente as atividades a serem executadas durante um período estimado. As tabelas 1 a 4, e respectivas figuras, são relativas ao cronograma estipulado para o desenvolvimento do software.

Tabela 1 - Cronograma de desenvolvimento da etapa I

	Nome da tarefa ▾	Duração ▾	Início ▾	Término ▾
1	Etapa I	21 dias	Sex 01/08/14	Sex 29/08/14
2	Efetuar o levantamento de requisitos.	16 dias	Qui 07/08/14	Qui 28/08/14
3	Efetuar a modelagem dos casos de uso.	5 dias	Sex 22/08/14	Qui 28/08/14
4	Pesquisar e definir as ferramentas mais adequadas para desenvolver o software.	2 dias	Seg 25/08/14	Ter 26/08/14
5	Definição e elaboração do cronograma de execução do projeto.	1 dia	Qui 28/08/14	Qui 28/08/14

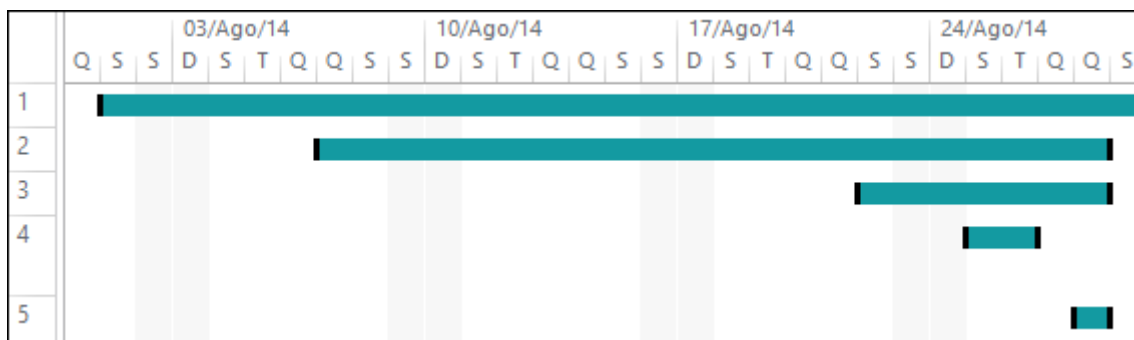


Figura 13 - Diagrama de Gantt referente a tabela 1

Tabela 2 - Cronograma de desenvolvimento da etapa II

	Nome da tarefa ▼	Duração ▼	Início ▼	Término ▼
1	Etapa II	16 dias	Sáb 30/08/14	Sex 19/09/14
2	Efetuar projeto sobre possibilidade/viabilidade de construir o software reutilizando padrões e componentes já existentes.	2 dias	Sáb 30/08/14	Seg 01/09/14
3	Efetuar projeto sobre possibilidade/viabilidade de construir o software com arquitetura distribuída.	3 dias	Seg 01/09/14	Qua 03/09/14
4	Efetuar projeto sobre possibilidade/viabilidade de construir o software com arquitetura orientada a serviços.	3 dias	Qua 03/09/14	Sex 05/09/14
5	Efetuar projeto sobre possibilidade/viabilidade de construir o software programado para ter manutenção facilitada.	3 dias	Sex 05/09/14	Ter 09/09/14
6	Efetuar projeto sobre possibilidade/viabilidade de construir o software programado para evoluir com facilidade.	3 dias	Ter 09/09/14	Qui 11/09/14
7	Efetuar a análise e modelagem de classes.	4 dias	Qui 11/09/14	Ter 16/09/14
8	Efetuar a análise e modelagem de sequência.	4 dias	Ter 16/09/14	Sex 19/09/14

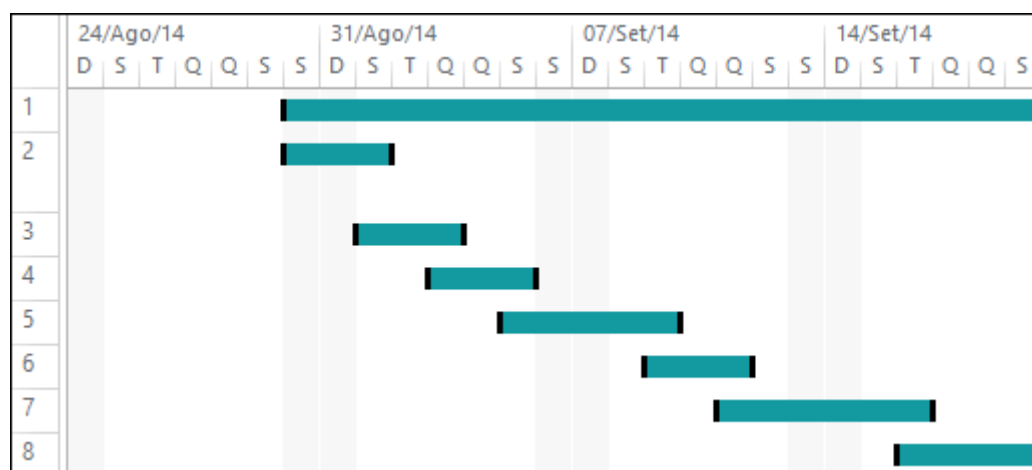


Figura 14 - Diagrama de Gantt referente a tabela 2

Tabela 3 - Cronograma de desenvolvimento da etapa III

	Nome da tarefa	Duração	Início	Término
1	Etapa III	13 dias	Sáb 20/09/14	Ter 07/10/14
2	Efetuar projeto de Banco de Dados.	7 dias	Sáb 20/09/14	Seg 29/09/14
3	Efetuar projeto de casos de teste que serão utilizados na etapa de testes do software.	4 dias	Seg 29/09/14	Qui 02/10/14
4	Efetuar estudo sobre confiança e proteção esperada do software a ser desenvolvido.	4 dias	Qui 02/10/14	Ter 07/10/14

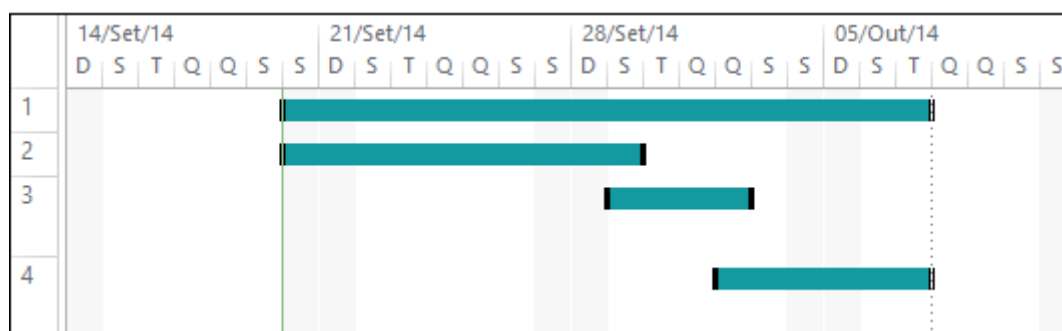


Figura 15 - Diagrama de Gantt referente a tabela 3

Tabela 4 - Cronograma de desenvolvimento da etapa IV

	Nome da tarefa	Duração	Início	Término
1	Etapa IV	33 dias	Qua 08/10/14	Sex 21/11/14
2	Efetuar a implementação.	21 dias	Qua 08/10/14	Qua 05/11/14
3	Efetuar testes usando os casos de teste.	4 dias	Qui 06/11/14	Ter 11/11/14
4	Efetuar implantação e validação com o usuário.	8 dias	Qua 12/11/14	Sex 21/11/14

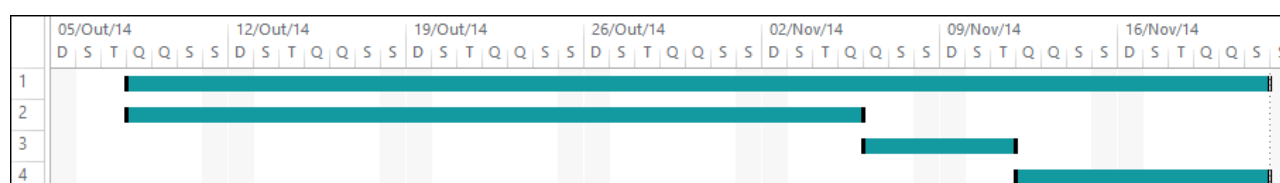


Figura 16 - Diagrama de Gantt referente a tabela 4

4.Projeto de padrões

No início do projeto foi apresentado pelo cliente um software similar ao requisitado, porém com uma grande dificuldade de utilização. É inviável tentar reutilizar parte do software apresentado, pois ele utiliza muitas fórmulas que não são necessárias para a criação do *SleepTree*.

Durante o desenvolvimento do software *SleepTree* será utilizado o padrão *Model-View-Controller* (MVC), que é uma arquitetura ou padrão que lhe permite dividir as funcionalidades de seu sistema em camadas, essa divisão é realizada para facilitar resolução de um problema maior. Onde possuímos três camadas básicas, e cada uma delas, com suas características e funções bem definidas para facilitar o desenvolvimento. Com a utilização

do padrão MVC, fica mais fácil a criação de uma atualização do software para rodar em diferentes plataformas, como por exemplo a plataforma Android[14] [15].

5. Estudos e Projeto da Arquitetura do Software

- **Arquitetura Distribuída**

Praticamente todos os grandes sistemas computacionais atualmente são sistemas distribuídos. Um sistema distribuído é um sistema que envolve vários computadores, em contraste com sistemas centralizados, em que todos os componentes do sistema executam em um único computador. Tanenbaum e Van Steen [12] definem um sistema distribuído como: Uma coleção de computadores independentes que aparece para o usuário como um único sistema coerente. “Um sistema distribuído permite o compartilhamento de recursos de hardware e software tais como discos impressoras, arquivos e compiladores que estão associados em computadores em uma rede”.

- **Arquitetura Orientada a Serviços**

As arquiteturas orientadas a serviços são uma forma de desenvolvimento de sistemas distribuídos em que os componentes de sistema são serviços autônomos, executando em computadores geograficamente distribuídos. Protocolos-padrão baseados em XML, SOAP e WSDL foram projetados para oferecer suporte a comunicação de serviço e a troca de informações Sommerville [1].

Nenhuma das arquiteturas será aplicada pois o software é simples, com apenas um usuário, ele não necessita de compartilhamento de dados no momento, futuramente alguma forma de arquitetura distribuída poderá ser aplicada no software, como por exemplo de aplicação, uma cooperativa que possui várias áreas, e nestas áreas um usuário que alimentará o banco de dados da cooperativa, e para uma melhor distribuição e interpretação dos dados um servidor poderá ser utilizado.

6. Estudos e projetos sobre manutenibilidade e evolução

A evolução é essencial, pois podem surgir novos requisitos; partes do software podem ser alteradas para a correção de erros ou para que o software se adapte a alterações no hardware ou software.

A manutenção de software é o processo geral de mudança em um sistema depois que ele é liberado para uso. As alterações feitas no software podem ser simples mudanças para correção de erros de codificação, até mudanças mais extensas para correção de erros de projeto ou melhorias significativas para corrigir erros de especificação ou acomodar novos requisitos.

O processo de evolução do software é dirigido pelas solicitações de mudança e inclui a análise do impacto da mudança, o planejamento de *release* e implementação da mudança Sommerville [1].



Para que haja uma boa evolução e manutenção, os seguintes pontos foram analisados:

- Software deve ser construído com a linguagem de programação JAVA que possui grandes vantagens, como portabilidade, orientação a objetos, recursos de rede, segurança.
- O software deve ser bem documentado e estruturado.
- Número e a complexidade das interfaces de sistema devem ser reduzidos.

- Estabilidade da equipe.

7. Previsões das Revisões e Inspeções

Inspeções centram-se principalmente no código-fonte de um sistema, mas qualquer representação legível do software, como seus requisitos ou modelo de projeto, pode ser inspecionado[1].

No decorrer do desenvolvimento deste trabalho já foi feita algumas inspeções, como por exemplo nos casos de uso que foram feitas algumas alterações. Agora que construímos os diagramas de sequência e de classe já é possível realizar novas inspeções no decorrer do projeto.

Podemos demonstrar aos clientes quais foram as funcionalidades que entendemos pela elicitação de requisitos e demonstramos através dos diagramas de casos de uso, classe e sequência para ver se estamos chegando em uma ideia comum. Caso o cliente perceba que estamos fazendo algo que não é necessário, ou que entendemos algo errado, ou que esquecemos de algum requisito então podemos inspecionar novamente nossos diagramas e documentos para atualizar de acordo com o que está sendo requisitado.

Além de consultar com os clientes para a identificação de erros, também é interessante sempre dar uma lida na documentação e nos diagramas para ver se não detectamos nenhuma falha.

Assim que começarmos a implementação do software poderemos também inspecionar o código-fonte do sistema para identificar possíveis erros.

8. Banco de Dados

SQLite é uma biblioteca em linguagem C que implementa um banco de dados SQL embutido. Programas que usam a biblioteca SQLite podem ter acesso a banco de dados SQL sem executar um processo SGBD separado [13].

SQLite não é uma biblioteca cliente usada para conectar com um grande servidor de banco de dados, mas sim o próprio servidor. É uma ferramenta que permite **com** que desenvolvedores possam armazenar os dados de suas aplicações em tabelas e manipular esses dados através de comandos SQL sem que seja preciso acessar um SGBD [13].

Na prática, o SQLite funciona como um “mini-SGBD”, capaz de criar um arquivo em disco e ler e escrever diretamente sobre este arquivo. O arquivo criado possui a extensão “.db” e é capaz de manter diversas tabelas. Uma tabela é criada com o uso do comando CREATE TABLE da linguagem SQL. Os dados das tabelas são manipulados através de comandos DML (INSERT, UPDATE e DELETE) e são consultados com o uso do comando SELECT assim como na linguagem SQL [13].

• Características do SQLite [13]:

- Software gratuito, multiplataforma, desenvolvido em C padrão (ANSI). Atualmente está na versão 3.8.7.
- Todo o banco de dados é guardado localmente (junto com a aplicação), em um único arquivo que possui a extensão “.db”. A base de dados pode ter tamanho superior a 2 terabytes.
- Não necessita de instalação, configuração ou de administração. Você não precisa ser um DBA para dominar o uso do SQLite.
- Suporta a maior parte do SQL 92.

- Não deve ser usado nos seguintes casos: aplicações de **alta concorrência** e sistemas ou aplicações web de porte muito grande.

• Considerações Finais

Como o SQLite não exige uma configuração de banco de dados no computador do usuário, acaba se tornando mais prático, pois assim que o usuário instalar o software Sleep-Tree em seu computador o banco de dados já estará configurado e pronto para uso sem que o cliente precise se preocupar com qualquer configuração.

8.1. Projeto do Banco de Dados

O banco de dados foi gerado com o relacionamento de quatro tabelas: Área, Temperatura, Unidade e Tipo. Na Figura 17 pode se ver os relacionamentos e os campos de cada tabela.

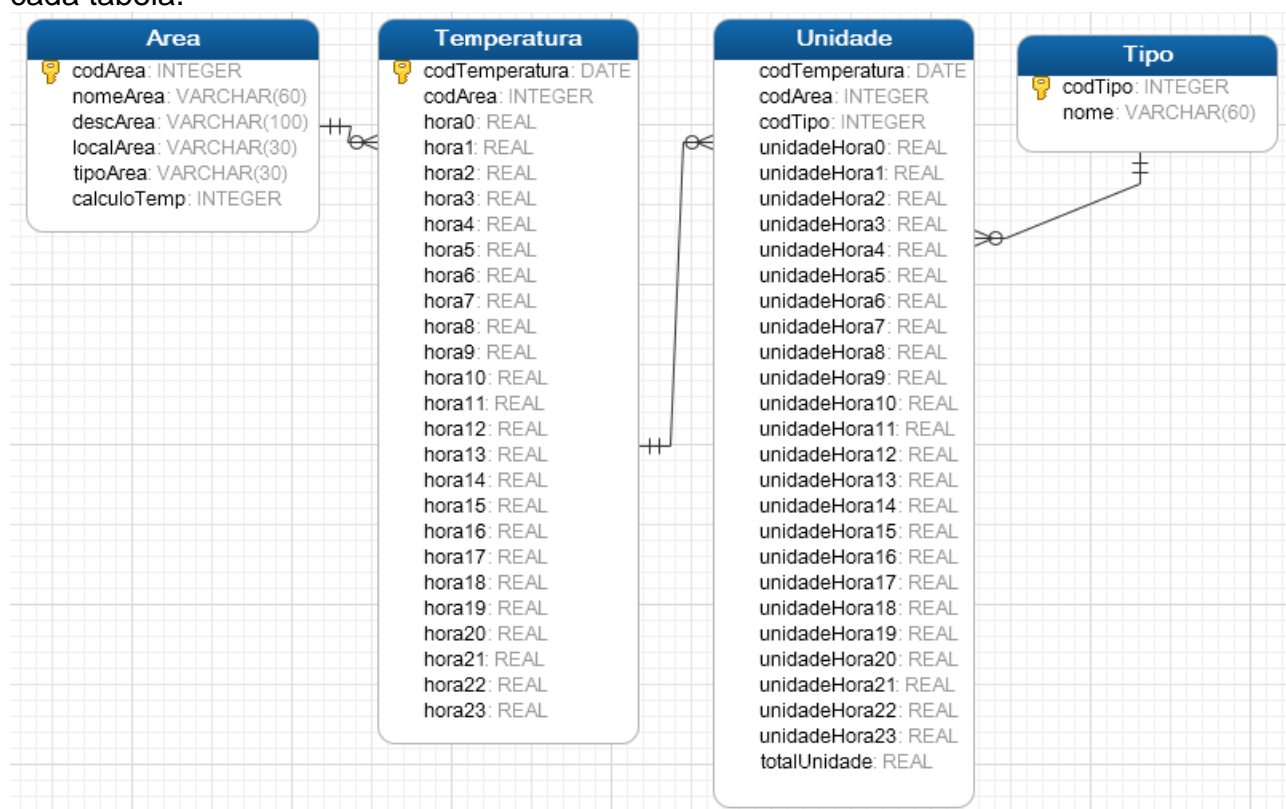


Figura 17 - Diagrama do Banco de Dados do SleepTree.

As tabelas 5 a 8 são relativas as descrições das tabelas da figura 17:

- **Área:** Esta tabela irá armazenar as áreas cadastradas pelo usuário, são armazenados seis campos. A explicação de cada campo está apresentada na Tabela 5.

Tabela 5 – Descrição de cada campo da tabela Área

Campo	Descrição
codArea	Representa a chave primária da tabela Área, cada área possui um código único

	<p>e será utilizado para representar determina área.</p> <p>Este campo é obrigatório e auto incremental.</p>
descArea	<p>Neste campo será armazenado uma breve descrição da área. Exemplo: “Chácara do Avô”.</p> <p>Campo opcional.</p>
localArea	<p>Neste campo será armazenado o local da área. Exemplo: “Guarapuava/PR”.</p> <p>Campo opcional.</p>
tipoArea	<p>Campo utilizado para armazenar a espécie da árvore cultivada na área. Exemplo: “Laranjeira”.</p> <p>Campo opcional.</p>
calculoTemp	<p>Quando o usuário registra uma nova área no programa ele deve escolher como o cálculo de interpolação será feito, com duas temperaturas (máxima e mínima) ou com três temperaturas (máxima, mínima e das 21 horas).</p> <p>Este campo irá armazenar um número inteiro que será o número dois ou três, representando a escolha do usuário.</p> <p>Campo obrigatório.</p>

- Temperatura:** Esta tabela irá armazenar a temperatura das vinte e quatro horas do dia. O usuário irá inserir apenas duas ou três temperaturas, de acordo com a decisão que ele tomar na hora de cadastrar uma nova área, as outras temperaturas do dia serão calculadas através do método de interpolação e cadastradas no banco pelo próprio programa. A tabela temperatura é referenciada pela sua chave primária codTemperatura do tipo DATE, pois o usuário deverá inserir as temperaturas apenas uma vez por dia, assim nunca repetindo a mesma data na chave codTemperatura. A explicação de cada campo da tabela Temperatura está apresentada na Tabela 6.

Tabela 6 – Descrição de cada campo da tabela Temperatura

Campo	Descrição
codTemperatura	Chave primária da tabela Temperatura, do tipo DATE, impedindo que o usuário insira mais de uma vez as temperaturas de um mesmo dia. Campo obrigatório, o usuário não precisará escrever a data, na janela de inserir as temperaturas irá possuir um mini calendário para que o usuário possa selecionar o dia, assim impedindo-o que insira valores inválidos.
codArea	Chave estrangeira da tabela Área, serve para relacionar a tabela Temperatura com a tabela Área. Campo obrigatório.
hora0, hora1, ..., hora23	Estes são os campos que irão armazenar a temperatura de todas as horas do dia, para que na hora de gerar relatórios de temperaturas o programa possa obter estes dados de forma mais rápida. Campos opcionais.

- **Tipo:** Está tabela irá armazenar os modelos utilizados para o cálculo de unidades de frio. Como foi definido nos requisitos do software, será utilizado três modelos: UTAH, Carolina do Norte, e um modelo padrão. Estes dados já estarão cadastrados no banco e não poderão ser alterados pelos usuários. Segue a explicação dos campos na Tabela 7.

Tabela 7 – Descrição de cada campo da tabela Tipo

Campo	Descrição
codTipo	Chave primária da tabela Tipo. Campo obrigatório e auto incremental.
Nome	Campo para armazenar o nome do Tipo. Exemplo: “Carolina do Norte”. Campo obrigatório.

- **Unidade:** Esta tabela irá armazenar as unidades de frio obtidas em cada hora do dia e o total obtido no final do dia. Esta tabela não será preenchida pelo usuário e sim pelo próprio software, através das temperaturas inseridas pelo usuário e as calculadas pela interpolação será calculada as unidades de frio de cada hora e depois somado todas para apresentar o total. Todo dia será preenchida três novas tabelas de Unidade, pois será registrado uma tabela de Unidade para cada codTipo da tabela Tipo. Segue a explicação dos campos na Tabela 8.

Tabela 8 – Descrição de cada campo da tabela Unidade

Campos	Descrição
codTemperatura	Chave estrangeira de Temperatura. Uma tabela de temperatura pode ter até três tabelas de Unidade enquanto que uma tabela de Unidade é representada por apenas uma tabela de Temperatura. Campo obrigatório.
codArea	Chave estrangeira de Área. Uma área pode possuir diversas tabelas de Unidade porém uma tabela de Unidade representa os dados de apenas uma área de um determinado dia. Campo obrigatório.
codTipo	Chave estrangeira de Tipo. Todos os dias precisa ser gerado as unidades de frio nos três modelos especificados pelo usuário que são: UTAH, Carolina do Norte e Padrão. Então todos os dias em que o usuário cadastrar as temperaturas será gerado três novas tabelas de Unidade representando cada tipo. Campo obrigatório.
unidadeHora0, ..., unidadeHora23	Estes campos armazenarão quantas unidades de frio foi registrada em cada hora do dia. Campos opcionais.

totalUnidade	<p>Campo que armazena o total de unidades de frio registradas no dia.</p> <p>Campo opcional.</p>
--------------	--

9. Projeto dos casos de testes

O teste é destinado a mostrar que um programa faz o que é proposto a fazer e para descobrir do programa antes do uso[1].


O projeto de casos de testes envolve entradas e saídas usadas para testar o sistema. A meta do projeto de casos de teste é criar um conjunto de testes que sejam eficazes em validação e teste de defeitos. 

Tabela 9 – Descrição dos casos de testes

ID	Módulo	Descrição	Roteiro	Resultado esperado
1	Áreas	Verificar a Validação dos campos, operação sem sucesso	Deixar todos os campos em branco e tentar criar uma área	Exibir mensagem de erro, dizendo quais são os campos obrigatórios.
2	Áreas	Verificar a Validação dos campos, operação com sucesso	Inserir as informações necessárias em todos os campos	Exibir mensagem avisando o sucesso da operação
3	Áreas	Inserção de áreas já existentes	Inserir uma área com o nome de uma já existente.	Exibir mensagem de erro, dizendo que a área já foi criada anteriormente.
4	Temperaturas	Inserção de temperaturas em uma nova área.	Selecionar uma área recém criada, para inserção de temperaturas	O programa deve detectar se a área é nova, e assim, liberar o campo 21a (21 horas do dia anterior)
5	Temperaturas	Inserção de temperaturas em uma área já existente	Selecionar uma área que recebeu a inserção de temperaturas.	O programa deve detectar que a área já possui temperaturas anteriores e bloquear o campo 21a.
6	Temperaturas	Verificar validação dos campos de temperaturas, operação sem sucesso.	Deixar os campos de temperaturas obrigatórios em branco e clicar no botão calcular.	Exibir mensagem de erro dizendo que existem campos em branco
7	Temperaturas	Verificar validação dos campos de temperaturas, operação com sucesso.	Preencher todos os campos de temperaturas e clicar no botão calcular	Exibir mensagem mostrando que a operação foi realizada com sucesso
8	Temperaturas	Realizar a verificação caso dados inválidos forem inseridos.	Inserir em um campo de temperatura dados do tipo char e clicar em calcular.	Exibir mensagem de erro dizendo que os tipos de dados devem ser numéricos
9	Temperaturas	Inserção de temperaturas em uma nova área.	Selecionar uma área e verificar se os campos de temperatura ativados correspondem a escolha do usuário na hora da criação da área (calculado com duas ou três temperaturas).	Que os campos ativados correspondam a opção que o usuário escolheu na hora em que criou uma nova área.

10. Confiança e proteção do software

- **Confiança**

Confiança de um sistema é uma propriedade agregada que leva em conta segurança, confiabilidade, disponibilidade e proteção e outros atributos do sistema. A confiança de um sistema reflete o quanto os usuários podem confiar nele[1].

Atualmente, a confiança dos sistemas costuma ser mais importante que sua funcionalidade, algumas dessas razões em relação ao **software** são:

- **Rejeição:** Se o usuário perceber que um sistema é não confiável ou não protegido, ele se **recusará** a usá-lo.
- **Custos de falhas mediano:** O erro nos cálculos das unidades de frio em plantações pode acarretar muito desperdício de recursos, **como por exemplo se aplicação de produtos agrícolas.**
- **Sistemas não confiáveis podem causar perda de informações:** Os dados são extremamente importantes logo a perda desses dados podem comprometer toda a funcionalidade do software, assim os cálculos das unidades de frio não serão possíveis.

Ao desenvolver o Sleeptree foram analisados os seguintes fatos: falha de hardware, software e sistema operacional. Geralmente essas falhas são inter-relacionadas. Um componente de hardware falho pode significar que os operadores de sistema precisam lidar com uma situação inesperada.

Para garantir um desenvolvimento de um software confiável foram garantidos os seguintes fatos:

- Seja evitada a introdução de erros acidentais no sistema durante a especificação e o desenvolvimento de software.
- Sejam projetados processos de verificação e validação, eficazes na descoberta de erros residuais que afetam a confiança.
- O sistema implementado e seu software de suporte sejam configurados corretamente para seu ambiente operacional.

Técnicas de desenvolvimento foram usadas para minimizar a possibilidade de erros humanos ou enganos antes que eles resultem na introdução de defeitos de sistema, e para assegurar que os defeitos em um sistema não resultam em erros de sistema ou erros que resultem em falhas do sistema.

- **Proteção**

A proteção é um atributo do sistema que reflete sua capacidade de se proteger de ataques externos, sejam acidentais ou deliberados. Esses ataques são possíveis porque a maioria dos computadores de uso geral está em rede e é, portanto, acessível a estranhos. Para alguns sistemas, a proteção é a dimensão mais importante da confiança de sistema [1]. Sistemas militares, sistemas de comércio eletrônico, por exemplo, devem ser projetados de modo a alcançar um elevado nível de proteção.

Existem três principais tipos de ameaças à proteção, ameaças à confidencialidade do sistema e seus dados, ameaças à integridade do sistema e seus dados e ameaças à

disponibilidade do sistema e seus dados

A ameaça à integridade do sistema e seus dados foi a mais considerável no desenvolvimento pois essas ameaças podem danificar o software ou corromper seus dados, como o software será implantado em local que não possui acesso à internet e os dados apenas são **uteis** para os usuários, ataques externos foram descartados.

Sem um nível razoável de proteção, não podemos estar confiantes quanto à disponibilidade, à confiabilidade e à segurança de um sistema. Erro no desenvolvimento de um sistema podem causar brechas de proteção[1].

Conclusão

O desenvolvimento de software é um processo dividido em uma série de etapas. Nesta etapa foram adicionadas atualizações no cronograma, o banco de dados e seu projeto, casos de testes e estudo sobre confiança e proteção do software. Observamos que o padrão MVC é útil, pois o software futuramente poderá receber uma atualização para ser executado em outra plataforma.

Referências

- [1] SOMMERVILLE, I., Engenharia de Software, 9ª Edição. São Paulo: Pearson – Addison Wesley, 2011.
- [2] Netbeans, Disponível em <<https://netbeans.org/>>. Acesso em 29 de agosto 2014.
- [3] BrModelo, Disponível em <<http://sis4.com/brModelo/>>. Acesso em 29 de agosto 2014.
- [4] Microsoft Visio, Disponível em <<http://office.microsoft.com/pt-br/visio/>>. Acesso em 29 de agosto 2014.
- [5] MySQL Workbench, Disponível em <<http://www.mysql.com/products/workbench/>>. Acesso em 29 de agosto 2014.
- [6] Postgree, Disponível em <<http://www.postgresql.org/>>. Acesso em 29 de agosto 2014.
- [7] Microsoft Project Manager, Disponível em <<http://office.microsoft.com/pt-br/project/>>. Acesso em 29 de agosto 2014.
- [8] Microsoft Office Word, Disponível em <<http://office.microsoft.com/pt-br/word/>>. Acesso em 29 de agosto 2014.
- [9] CodeBlocks, Disponível em <<http://www.codeblocks.org/>>. Acesso em 29 de agosto 2014.
- [10] Google Drive Docs, Disponível em <<https://drive.google.com/>>. Acesso em 29 de agosto 2014.
- [11] Skype, Disponível em <http://www.skype.com/pt_BR/>. Acesso em 29 de agosto 2014.
- [12] TANENBAUM, A. S.; STEEN, M. V., Distributed Systems: Principles and Paradigms, 2ª Edição. Pearson Prentice Hall, 2007.
- [13] SQLite. About SQLite. Disponível em: <<http://www.sqlite.org/about.html>>. Acesso em: 20 de out. de 2014.
- [14] Model-View-Controller (MVC). Disponível em: <<http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm>>. Acesso em: 20 de out. de 2014
- [15] MVC – Model View Controller. Disponível em:<<http://www.caelum.com.br/apostila-java-web/mvc-model-view-controller/#9-8-fazendo-a-logica-para-listar-os-contatos>>. Acesso em: 20 de out. de 2014.