DEPARTAMENTO DE COMPUTAÇÃO

D E C O M

TerraME *Observer*

Ferramentas para visualização gráfica e análise da dinâmica de modelos ambientais espacialmente-explícitos

U F O P

UNIVERSIDADE FEDERAL DE OURO PRETO

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

TerraME *Observer*
Ferramentas para visualização gráfica e análise da dinâmica de modelos ambientais espacialmente-explícitos

Antônio José da Cunha Rodrigues

Orientador: Prof. Dr. Tiago Garcia de Senna Carneiro

OURO PRETO, MG – BRASIL
DEZEMBRO DE 2008

FOLHA DE APROVAÇÃO DA BANCA EXAMINADORA

TerraME *Observer*
Ferramentas para visualização gráfica e análise da dinâmica de modelos ambientais espacialmente-explícitos

Antônio José da Cunha Rodrigues

Apresentada ao Departamento de Computação do Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto como requisito parcial da disciplina CIC391 – Monografia, do curso de Bacharelado em Ciência da Computação.

Aprovada por:

_____

Prof. Dr. Tiago Garcia de Senna Carneiro
Doutor pelo Instituto Nacional de Pesquisas Espaciais
Departamento de Computação – UFOP

_____

Prof. Dr. Ricardo de Oliveira Duarte
Doutor pela Institut National Polytechnique de Grenoble – França
Departamento de Computação – UFOP

_____

Prof. Dr. Frederico Gadelha Guimarães
Doutor pela Universidade Federal de Minas Gerais
Departamento de Computação – UFOP

OURO PRETO, MG – BRASIL
DEZEMBRO DE 2008

Resumo

Este trabalho propõe o desenvolvimento do módulo TerraME *Observer*, que será adicionado à arquitetura TerraME, para dotá-la de serviços para a análise e visualização de modelos ambientais. Tal ferramenta trará agilidade nas análises de resultados provenientes do ambiente TerraME permitindo ao modelador monitorar, em tempo de simulação, toda a dinâmica do modelo em estudo. Para prover tais serviços, foi desenvolvida uma plataforma de *software* com vasto uso de padrões de projeto e mecanismos de serialização e desserialização, desta forma, dados provenientes da camada de aplicação (camada superior) produzidos pelo modelador, puderam ser utilizados em camadas inferiores para a geração de observadores dinâmicos. Os resultados, mesmo que iniciais, demonstram a potencialidade dos mecanismos desenvolvidos e transparecem o principal objetivo deste sistema, a redução do tempo de análise e o acompanhamento *on-line* da simulação do modelo em estudo.

Agradecimentos

A Deus e a Virgem Maria, pela força nesta caminhada.

A minha família, pelo apoio e incentivo durante esse difícil caminho.

Ao professor Tiago Carneiro, pela orientação e ensinamentos transmitidos e pela confiança em minha capacidade.

Aos professores do DECOM, pelo ensinamento e por mostrarem qual caminho percorrer.

Aos amigos da UFOP, por compartilharem o tempo, a amizade e o conhecimento adquirido.

A todos os que, de alguma forma, contribuíram para a minha formação.

E a minha namorada Ivanildes, pelo carinho, compreensão, companheirismo e apoio.

Sumário

Lista de Exemplos

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

API	–	*Application Programming Interface*
CA	–	*Cellular Automata*
FIOCRUZ	–	Fundação Oswaldo Cruz
GUI	–	*Graphics User Interface*
INPE	–	Instituto Nacional de Pesquisas Espaciais
LUCC	–	*Land-use and land-cover change*
Nested-CA	–	*Nested Cellular Automata*
PUC	–	Pontifícia Universidade Católica
QT	–	QT*oolkit* – Trolltech
QWT	–	QT *Widgets for Technical Applications*
SIG	–	Sistemas de Informação Geográfica
UML	–	*Unified Modeling Language*

1 – Introdução

Neste capítulo, apresentaremos o desafio da análise de resultados dos modelos simulados no ambiente TerraME, a dificuldade de visualização e acompanhamento da dinâmica do modelo, em seguida, descreveremos o objetivo e a estruturação deste trabalho.

1.1 – O desafio da análise de resultados

O ambiente de modelagem TerraME é uma arquitetura de *software* destinada ao desenvolvimento de modelos dinâmicos espacialmente-explícitos (PARKER *et al*, 2002), que vem sendo utilizada, por exemplo, na modelagem e simulação de processos de mudanças de uso e de cobertura do solo (do inglês, Land-use and land-cover change – LUCC) para toda região amazônica (VELDKAMP *et at*, 1996), no âmbito do projeto GEOMA [www.geoma.lncc.br], e no desenvolvimento de modelos epidemiológicos, em parceria com a FIOCRUZ, como nos casos do controle da Dengue nas cidades do Rio de Janeiro - RJ, e Recife - PE.

Na maioria dos experimentos conduzidos nesses projetos, tanto a prototipagem de modelos quanto a análise dos resultados por eles gerados são atividades realizadas de maneira ineficiente e artesanal e, portanto, demandam uma enorme quantidade de tempo. Durante a simulação de um sistema, o modelador poderá, em alguns casos, se deparar com a necessidade de verificar a evolução de algum agente desse modelo e, inevitavelmente, ele precisará esperar o término dessa simulação para que possa analisar sua evolução e assim, corrigir algum eventual problema encontrado.

Portanto, trata-se de um problema de grande importância e interesse científico, pois, conforme descrito anteriormente, o desafio será minimizar o tempo gasto para simular uma determinada atividade, facilitar a percepção inicial de falhas na definição do modelo estudado além de permitir o monitoramento e a visualização da dinâmica da simulação.

1.2 – Estrutura do trabalho

Este trabalho foi estruturado em oito capítulos, incluindo essa introdução.

No segundo capítulo, apresentamos os objetivos deste trabalho.

No posterior, descrevemos a justificativa para o desenvolvimento deste trabalho.

No capítulo seguinte, apresentamos os fundamentos teóricos empregados, a bibliografia consultada na confecção deste projeto e trabalhos correlatos encontrados na literatura.

No capítulo quinto, demonstramos os materiais e métodos utilizados nessa produção.

No capítulo posterior, descrevemos o sistema computacional desenvolvido.

No sétimo capítulo, apresentamos os resultados obtidos utilizando o sistema desenvolvido.

No último capítulo, descrevemos a conclusão deste trabalho e apresentamos os trabalhos futuros.

2 – Objetivos

2.1 – Objetivo geral

O presente trabalho teve por objetivos, propor o desenvolvimento de uma ferramenta para auxiliar a análise dos resultados provenientes da simulação e permitir o acompanhamento o*n-line* da evolução dos modelos em estudo. A ferramenta terá o intuito de reduzir o tempo da simulação no sentido de que não mais será necessário aguardar o término dessa para eventuais correções, pois durante a sua execução, o modelo poderá ser alterado, reduzindo o tempo de análise, correção e re-execução do modelo. Além de deixar a cargo do modelador a verificação, durante a execução, de toda simulação.

2.2 – Objetivo específico

Desenvolver uma arquitetura de *software* onde, informações (dados) do modelo em estudo definidos na camada de aplicação são interpretados e utilizados em camadas inferiores, através de mecanismos de serialização e desserialização, para a geração de tabelas, gráficos e imagens dinâmicos.

3 – Justificativas

O processo de análise de uma simulação, desenvolvido através dos resultados gerados pelo ambiente de modelagem TerraME, é a parte da modelagem na qual temos uma maior demanda de tempo, tal fato, justifica o desenvolvimento desse trabalho.

Atualmente, a plataforma TerraME não permite o acompanhamento da simulação em tempo de execução, não possui ferramentas gráficas de auxílio à análise de resultados gerados e nem mecanismos de acompanhamento passo a passo da simulação, pois executa o modelo de uma só vez. Projetos como modelagem de mudança de uso e cobertura de solo, modelagem da dinâmica populacional do mosquito da dengue e dispersão da doença no Rio de Janeiro conduzida por pesquisadores da FIOCRUZ, INPE e UFOP, desenvolvidos sobre o ambiente de modelagem, têm muito a lucrar com as ferramentas produzidas neste trabalho. Tais ferramentas gráficas tornarão mais fáceis as análises de resultados de modelos desenvolvidos para os mais variados domínios de aplicação, além de permitir o acompanhamento o*n-line* de toda a dinâmica da simulação.

As pesquisas nesse sentido poderão levar ao desenvolvimento de novas técnicas computacionais e de modelagem que poderão ajudar a reduzir o tempo de análise e auxiliar no desenvolvimento de medidas de contingência para que seja possível evitar ou, ao menos, minimizar o impacto de desastres naturais ou epidemiológicos. Além de contribuir para a elaboração de prognósticos mais precisos acerca do comportamento das interações entre o homem e o ambiente.

4 – Revisão bibliográfica

Neste capítulo, apresentaremos os fundamentos teóricos, os trabalhos correlatos e a bibliografia utilizados na produção deste trabalho.

4.1 – Fundamentação teórica

É importante salientar o significado de um modelo e sua utilização no decorrer desse trabalho, ele é uma representação simplificada da realidade que utiliza artifícios físicos, matemáticos, computacionais, entre outros. Modelos ambientais reproduzem a realidade inerente ao meio ambiente, assim como as ações que modificam esse meio, já os modelos espacialmente-explícitos denotam a localização física do ambiente de estudo.

A programação orientada a objetos possibilitou que o desenvolvimento de *software* fosse feito em módulos, assim todas as funcionalidades dependentes ficam combinadas em uma única parte. Desta forma, tanto a manutenção quanto as melhorias no código se tornam locais, permitindo com que toda a estrutura desse módulo possa ser alterada preservando a comunicação com os demais módulos. A arquitetura em camada foi utilizada de forma análoga, camadas independentes formam a base do sistema, em seguida, as camadas dependentes são colocadas sobre a anterior. Assim, as camadas inferiores provêem os serviços básicos que oferecem suporte aos serviços das camadas superiores. Portanto, cria-se uma dependência vertical e uma independência horizontal entre os módulos.

4.2 – Trabalhos correlatos

Atualmente, temos na literatura, diversos ambientes de modelagem que permitem ao modelador visualizar os resultados dos modelos de estudo em gráficos, tabelas e imagens. Dentre eles, escolhemos os dois mais relevantes, o STELLA e o SWARM.

- **STELLA:** esta plataforma baseia-se na teoria de sistemas (ROBERTS *et al*, 1983), utiliza diagramas de fluxo e sistemas (retângulos) conectados por fluxos de energia para a criação de modelos. Para a visualização dos resultados, ele utiliza gráficos. Entretanto, não permite o acompanhamento o*n-line* da simulação do modelo estudado. Quanto aos gráficos, eles são estáticos, ou seja, não são alterados dinamicamente no decorrer da evolução da simulação.

- **SWARM:** esta plataforma, baseada na modelagem espacial dinâmica (MINAR *et al*, 1996), utiliza o conceito de *probes* que são responsáveis por exibir GUIs ao usuário, eles inspecionam a evolução do modelo, por exemplo, um gráfico dinâmico ou uma imagem espacial dinâmica. Permite que o modelador crie seu próprio *probe*. Contudo, ele exige do modelador conhecimentos avançados de programação, pois utiliza a linguagem *Objective-C* (baseada na linguagem C) e a cada alteração o modelador precisará re-compilar o modelo.

4.3 – Os padrões de projeto

Um padrão de projeto na computação descreve um problema e uma solução adotada para atacar esse problema, ele também define um nome para a apresentação desse padrão. A solução se torna um padrão, pois ela já foi utilizada com sucesso em diversos projetos sendo apenas adaptada para cada situação. Também identifica um cenário para o qual um determinado padrão deve ser adotado em detrimento a outro e quando devemos utilizar um conjunto de padrões para solucionarmos um problema (GAMMA *et al*, 1995).

Padrões de projeto são conceituados por Erich Gamma *et al*, como: "(...) são descrições de objetos e classes comunicantes que são customizados para resolver um problema geral de projeto num contexto particular."(GAMMA *et al*, 1995, p. 20)

4.3.1 – O padrão *Observer*

Esse padrão de comportamento é usado para manter a consistência entre objetos, quando o estado de um objeto *subject* (assunto) muda, ele notifica os demais objetos interessados nesta mudança, os observadores (GAMMA *et al*, 1995). Ele demonstra como podemos visualizar um mesmo dado de várias formas, porém sempre mantendo a consistência entre estas visualizações, pois se o dado mudar, o padrão se encarregará de manter todas estas apresentações coerentes com esta mudança.

4.3.2 – O padrão *Factory Method*

A escolha do nome de um padrão está relacionada à sua utilização. O *Factory Method* é um padrão de criação, ele define uma interface virtual e delega à subclasse quando e qual o objeto instanciar (GAMMA *et al*, 1995).

4.3.3 – O padrão *Bridge*

É um padrão comportamental, ele mantém a interface e a implementação em hierarquias de classes separadas de maneira que ambas possam evoluir de forma independente (GAMMA *et al*, 1995).

4.3.4 – O padrão *Template*

Também chamado de programação genérica, esse também é um padrão comportamental, pois a classe é construída de forma que o código comum a várias classes seja escrito apenas uma vez e o código variante entre elas seja definido apenas em suas subclasses (GAMMA *et al*, 1995).

5 – Materiais e métodos

Nesta seção, apresentaremos os sistemas computacionais utilizados e a metodologia empregada no processo de desenvolvimento deste projeto.

5.1 – Sistemas computacionais

Demonstraremos nesta subseção, os sistemas computacionais empregados para a produção do sistema.

5.1.1 – TerraME

O ambiente de modelagem *Terra Modeling Enviroment* – TerraME é um modelo computacional desenvolvido para avaliar o modelo de Autômatos Celulares Aninhados – Nested-CA (CARNEIRO, 2006), baseado em dois outros modelos, o de Autômatos Celulares e o de Agentes. O Nested-CA considera inúmeros níveis de escala onde cada uma delas esta associada a uma resolução espacial, analítica e temporal. Para armazenar e recuperar os resultados dos modelos dinâmicos espaciais, ou seja, os mapas que descrevem a distribuição espacial de um padrão ou de uma variável, o TerraME utiliza a biblioteca TerraLib. Essa biblioteca foi desenvolvida através da linguagem C++ para prover o acesso e gerenciamento de bancos de dados geográficos.

O *framework* TerraME, módulo escrito na linguagem C++, dispõe de classes e funções para a modelagem dinâmico-espacial e possui um interpretador que recebe o código fonte de um modelo escrito na linguagem de modelagem TerraME. Tal linguagem é uma extensão da linguagem LUA (IERUSALIMSCHY, 2006) e por meio desta, o interpretador TerraME lê, interpreta o modelo e faz chamadas às funções do *framework*. A Figura 1 mostra um esquema do interpretador do TerraME e seus serviços.

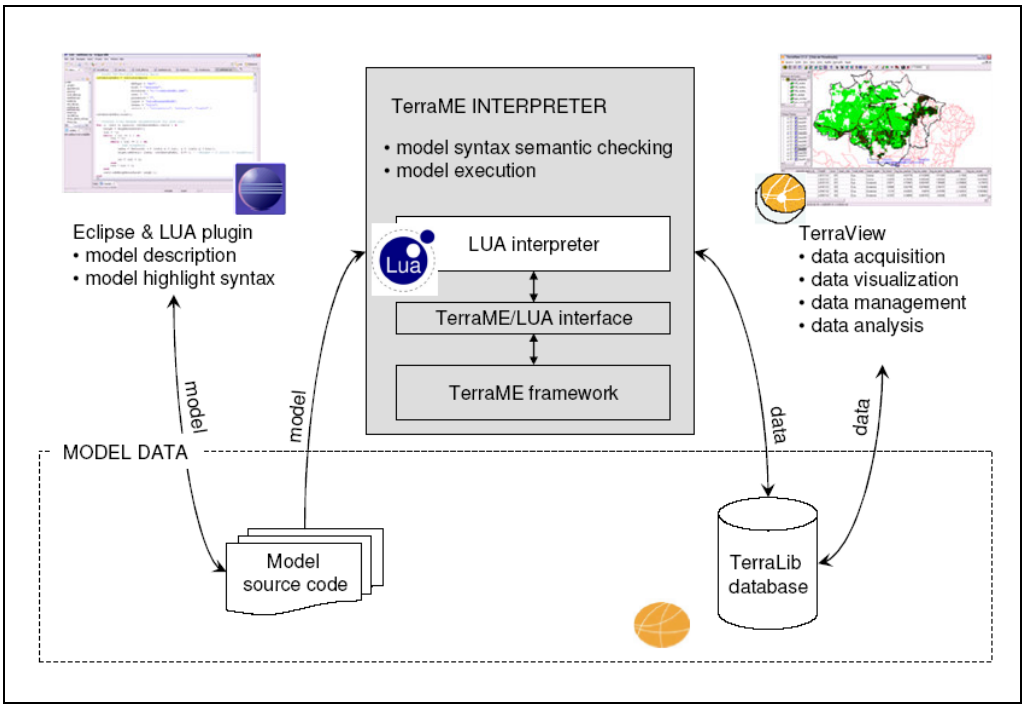

Figura 1 – Módulo TerraME e serviços (Fonte: CARNEIRO, 2006)

Para a elaboração do código do modelo, o pesquisador pode utilizar qualquer editor de texto puro, porém a plataforma de desenvolvimento Eclipse provê recursos que facilitam essa elaboração, como por exemplo, destaque da sintaxe e a facilidade de chamar o interpretador do *framework* TerraME dentro deste mesmo ambiente. O interpretador faz a verificação da sintaxe e da semântica do modelo, sendo também o responsável pela execução do modelo. A biblioteca TerraLib é usada para leitura das entradas do modelo e para a gravação dos resultados da simulação além de fornecer mecanismos para o gerenciamento de bancos de dados espaciais. Para a visualização, a análise e o gerenciamento dos dados é utilizado o *software* TerraView.

A Figura 2 descreve a arquitetura da plataforma de modelagem TerraME.

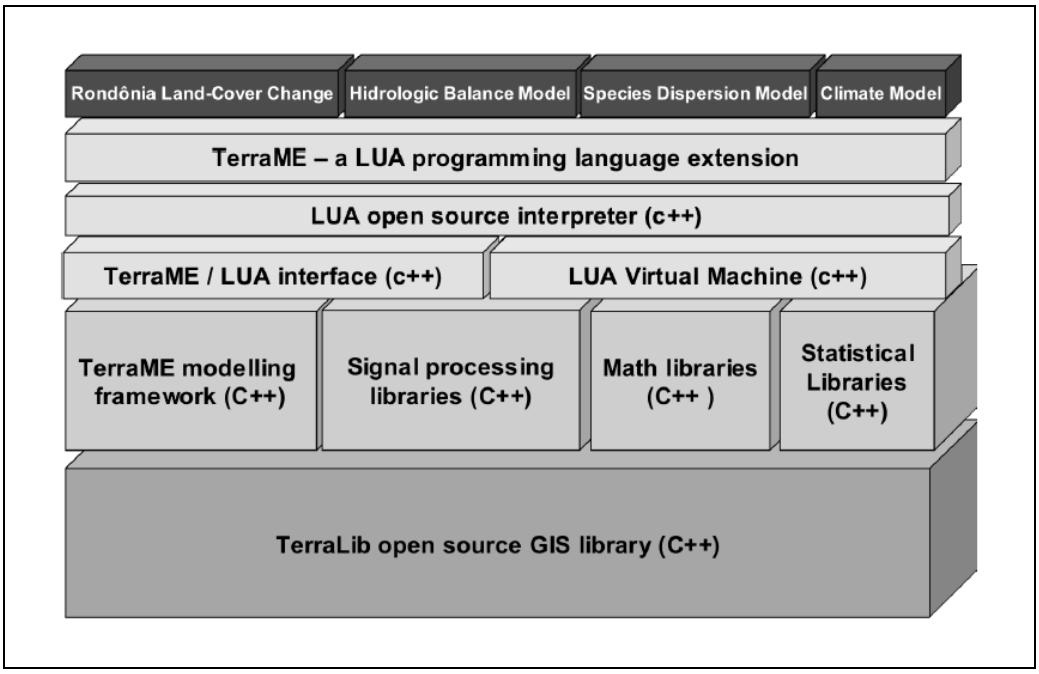

Figura 2 – Arquitetura TerraME (Fonte: CARNEIRO, 2006)

Na primeira camada, a TerraLib oferece sistemas de informação geográfica (SIG), gestão de dados espacial, serviços de análise e funções extras para a manipulação de dados temporais. Na segunda, o *framework* TerraME provê a simulação e os serviços de calibração e validação. Foi desenvolvido para ser independente da plataforma. Este *framework* pode ser utilizado diretamente para desenvolvimento de modelo, porém o desenvolvimento de modelos na linguagem C++ pode ser um desafio para modeladores, por isso, o TerraME proporciona uma linguagem de modelagem de alto nível. A terceira camada da arquitetura implementa a linguagem de modelagem TerraME e o ambiente de execução. O TerraME / LUA *interface* estende LUA com os novos tipos de dados espaciais de modelagem dinâmica e serviços para o modelo de simulação e avaliação. Para tornar possível utilizar o *framework* através do interpretador LUA, foi necessário exportar a API do TerraME para a API de LUA, assim o TerraME reconhece os tipos descritos no modelo. Através dessa exportação outras aplicações escritas em C ou C++ podem ter suas APIs exportadas para a linguagem LUA. A última camada é chamada camada de aplicação e inclui os modelos de usuário final.

O TerraME utiliza vários tipos de objetos (CARNEIRO, 2006), os principais estão descritos abaixo.

❖ **O tipo Escala**

É um contêiner para autômatos, espaços celulares e temporizadores. Os autômatos representam os atores e os processos envolvidos; os espaços celulares são representações de cada local e os temporizadores são responsáveis por definir a ordem da simulação. O tipo escala é o tipo mais alto na hierarquia do TerraME, assim ele pode conter todos os demais tipos definidos no *framework*.

❖ **O tipo Célula**

Célula, em TerraME, significa um local no espaço com propriedades, atributos e relações. Ela possui dois atributos principais o *latency* e o *past*. O atributo *latency* representa o estado atual da célula desde a sua última modificação, já o atributo *past* representa o último estado da célula, ele é uma cópia da célula em um determinado instante do passado.

❖ **O tipo Espaço Celular**

O espaço celular é um conjunto multivalorado de células e está associado a um banco de dados geográfico.

❖ **O tipo Vizinhança**

Toda célula possui algum vizinho, esse tipo representa a relação entre essa célula e cada um de seus vizinhos. Ele possui dois atributos: a célula, que representa cada um dos vizinhos e o peso, que demonstra a relação entre ambos.

❖ **O tipo Temporizador**

É o tipo responsável por controlar a simulação, ele trabalha em conjunto com o tipo escala, pois cada resolução possui necessariamente uma escala e um temporizador.

5.1.2 – A Linguagem LUA

É uma linguagem de *script* criada na PUC - Rio em 1993. Foi desenvolvida com o intuito de ser estendida (IERUSALIMSCHY, 2006), priorizando cinco itens: portabilidade, simplicidade, pequeno tamanho, "acoplabilidade" e eficiência (IERUSALIMSCHY, 2007).

- **Portabilidade**

Criada para ser utilizada em qualquer plataforma e também em microprocessadores embutidos.

- **Simplicidade**

Utiliza um único tipo de estrutura de dados (tabela) que implementa a maioria das estruturas de forma simples e eficiente. Um único tipo numérico o *double*.

- **Pequeno Tamanho**

Núcleo com menos de 200KB, tem uma interface bem definida e bibliotecas independentes. Pode ser inserida em aplicações com aumento mínimo no tamanho.

- **"Acoplabilidade"**

É uma biblioteca C, sua API é simples e bem definida: tipos simples, operações primitivas e modelo de pilha. Pode ser acoplada em qualquer linguagem, por exemplo, C, C++, Java, Perl, Fortran, entre outras.

- **Eficiência**

Mistura a simplicidade e algumas técnicas especiais, tais como, *meta-mecanismos* para a implementação de construções, de classes e herança. Estudos mostram LUA entre as mais rápidas no grupo de linguagens interpretadas com *tipagem* dinâmica.

LUA é uma linguagem de programação rápida, leve e principalmente poderosa, e há alguns anos é usada em larga escala no mercado. Sua escolha se deve ao fato de ser simples e de fácil compreensão além de possuir um grande poder de representação. Ela auxilia o modelador a gerar o código do modelo de forma mais legível.

5.1.3 – O *toolkit* QT

Desenvolvido na linguagem C++, pela Trolltech [www.trolltech.com], para a criação de interfaces gráficas (GUIs) independentes de plataforma. Nele foi utilizado o conceito de *signals* e *slots*, um paradigma simples, mas muito eficiente na criação de GUIs (BLANCHETTE *et al*, 2004). É uma implementação do padrão *Observer* que funciona da seguinte forma: quando ocorre alguma interação entre o usuário e a interface gráfica ou quando algum componente dessa interface teve seu estado modificado, é gerado um *signal* que está ligado a um método chamado *slot* responsável por tratar essa interação. Por exemplo, um botão possui um *signal* chamado *clicked()* associado a ele, quando o usuário clica nesse botão o sinal *clicked()* é emitido e, para tratar o efeito causado pelo clique, o *slot* correspondente é chamado. Alguns *slots* são utilizados como padrão, entretanto podem ser personalizados ou mesmo associar mais *slots* a um mesmo *signal*.

5.1.4 – O *framework* QWT

O QWT é um *framework opensource* [qwt.sourceforge.net] baseada no *toolkit* QT, o que a torna independente de plataforma. Ele provê, entre outros itens, recursos para a geração de gráficos.

5.2 – Métodos

5.2.1 – Processo de desenvolvimento

Foi utilizada a metodologia de desenvolvimento de *software* em espiral com prototipação de *releases* e versões. Assim, as etapas de concepção, projeto, implementação, teste e documentação estão sendo executados de forma cíclica e a cada ciclo uma nova versão documentada do aplicativo é obtida (Figura 3).

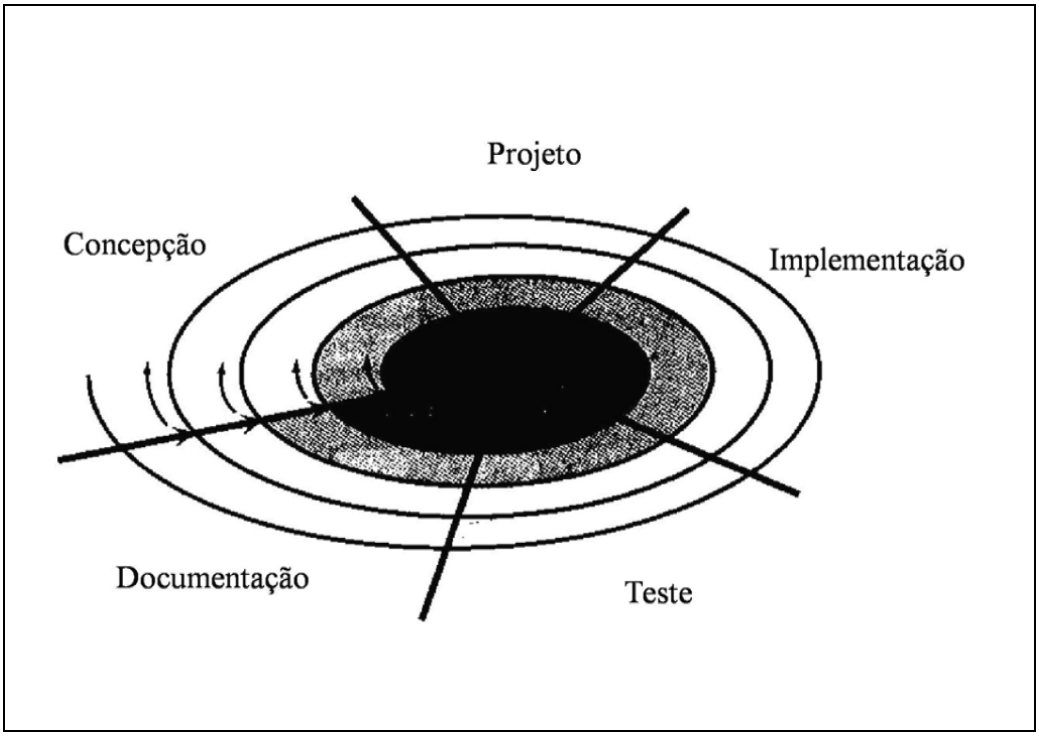

Figura 3 – Processo de Desenvolvimento de *Software*
(Fonte: PRESSMAN, 2005)

❖ **Concepção de software**

Nesta fase, são definidas as questões que o aplicativo TerraME *Observer* deve responder. Os trabalhos correlatos são investigados. As informações disponíveis são analisadas na esperança de que sejam identificados todos os requisitos de software. O documento de especificação será elaborado nesta etapa.

❖ **Projeto de software**

Nesta etapa, são projetados a arquitetura, os algoritmos, as estruturas de dados e as interfaces com o usuário do sistema TerraME *Observer*. Os diagramas de classe, de seqüência, de transição de estados e da metodologia UML serão utilizados para a documentação das decisões de projeto. O documento de projeto será elaborado nesta fase.

❖ **Implementação de software**

Nesta fase, cada módulo de software proposto na etapa anterior será implementado na linguagem de programação C++. O código fonte da arquitetura será documentado através do sistema de documentação DOXYGEN [www.doxygen.org].

❖ **Teste de software**

Nesta etapa, cada módulo de software implementado na fase anterior será testado e qualquer falha será analisada para posterior correção.

❖ **Documentação de software**

Nesta fase, são escritos artigos e relatórios sobre os avanços científicos e tecnológicos gerados pela pesquisa, o manual do usuário, o guia de programação e os tutorias do aplicativo TerraME *Observer*.

5.2.2 – Requisitos

Os requisitos descritos a seguir foram levantados durante a concepção, planejamento do projeto.

Requisitos de portabilidade

O aplicativo foi planejado para que possa ser utilizado em qualquer plataforma. Por isso esta sendo desenvolvido utilizando a linguagem de programação C++ e sua interface gráfica utilizando o *toolkit QT*.

Requisitos de Negócio

i. Variáveis do modelo poderão ser observadas, por gráficos dinâmicos, tabelas dinâmicas ou mapas dinâmicos, em tempo de simulação, ainda durante a simulação do modelo e após seu término.

ii. Um ou mais observadores (interfaces gráficas para visualização das variáveis do modelo) poderão ser associadas simultaneamente a uma única variável do modelo.

iii. Qualquer mudança em uma variável do modelo deve ser, de forma imediata e automática, refletidas nos observadores a ela associada.

iv. Os modelos já desenvolvidos em TerraME devem continuar funcionando sem qualquer alteração em seu código fonte.

v. A definição, instanciação e associação de observadores deverão ser realizadas por primitivas registradas na linguagem de modelagem TerraME (uma extensão da linguagem de programação LUA).

vi. O modelador poderá, a qualquer momento, interromper, finalizar, avançar e retroceder uma simulação para observar em detalhe sua dinâmica.

6 – O Sistema Desenvolvido

6.1 – Introdução

O sistema desenvolvido, chamado TerraME *Observer* cujo intuito é facilitar a confecção, o monitoramento e o auxílio nas análises dos resultados obtidos durante a simulação de modelos ambientais desenvolvidos sobre a ambiente TerraME. Este sistema foi produzido utilizando a linguagem C++ e o intenso uso de padrões de projeto.

6.2 – Arquitetura

O sistema foi desenvolvido em camadas, as inferiores provêem serviços às superiores, e em módulos, procedimentos similares ou dependentes ficam inclusos no mesmo módulo, assim produzimos um software fortemente conciso e fracamente acoplado, ou seja, dependência vertical entre camadas e independência entre módulos pertencentes à mesma camada. Na primeira camada temos, o *framework* TerraME responsável por efetuar a simulação. Na segunda, temos o TerraME *Observer* que agrega novos mecanismos enriquecendo o ambiente de simulação para que o modelador retire o máximo de desempenho deste novo *software* e, temos também, o *toolkit* QT responsável por toda a interface gráfica e o *framework* QWT responsável pela geração de todos os gráficos. Na terceira camada, temos a linguagem de modelagem TerraME, utilizada para o descrição dos modelos simulados e na última camada temos, os modelos do usuário final. A Figura 4 sintetiza toda a arquitetura desse aplicativo.

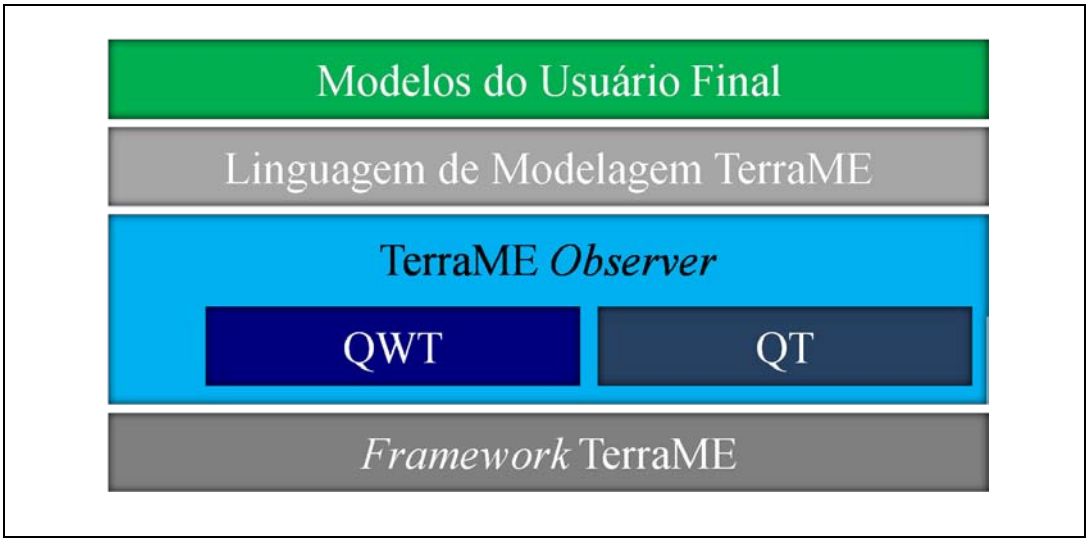

Figura 4 – Arquitetura TerraME *Observer*

6.3 – Diagrama de Classe

A Figura 5 apresenta o diagrama UML completo, ele mostra o relacionamento entre as classes do sistema. Para uma fácil visualização, a função da cada classe foi separada por uma cor característica. Os primeiros nós do diagrama, em tons de verde, representam os padrões de projeto utilizados (Figura 6). À esquerda, em tons de azul, temos as interfaces e em tons de amarelo, à direita, apresentamos as implementações. Abaixo, em vermelho, temos os modelos do usuário final.

As classes do padrão *Observer* estão representadas em verde limão, o principal padrão de projeto utilizado nesse trabalho, elas são classes abstratas e são desenvolvidas duas vezes, na interface e na implementação. Já as classes do padrão de projeto *Bridge*[1], estão verde claro, elas separam as classes de interfaces das classes de implementações, para que ambas possam evoluir sem que haja interferência de uma sobre a outra.

❖ As classes em tons de azul (Figura 7):

Essas classes, como já mencionado, são as interfaces, elas fazem a comunicação entre um objeto interface e um objeto implementação. Em azul claro, temos as interfaces do padrão *Observer*, através do diagrama pode-se perceber que essas classes herdam dos padrões *Bridge* e *Observer*. Em azul escuro, apresentamos a terceira camada na hierarquia de classes, essas são as interfaces dos tipos de objetos observáveis, tais como, espaço celular e célula. Nessa camada, também temos padrão de projeto *Factory Method* que irá fabricar os observadores de acordo com um tipo pré-definido, por exemplo: *TME_Graphic*, observador do tipo gráfico, *TME_Table*, observador do tipo tabela, *TME_CellSpace*, observador do tipo espaço celular. O Exemplo 1 demonstra o método de fabricação de um observador *TME_Graphic*.

[1] O padrão *Bridge*, como já mencionado, separa a interface de sua implementação, entretanto, o programador precisa desenvolver o código referente à interface que irá invocar, durante a execução, o código referente à implementação desse objeto.

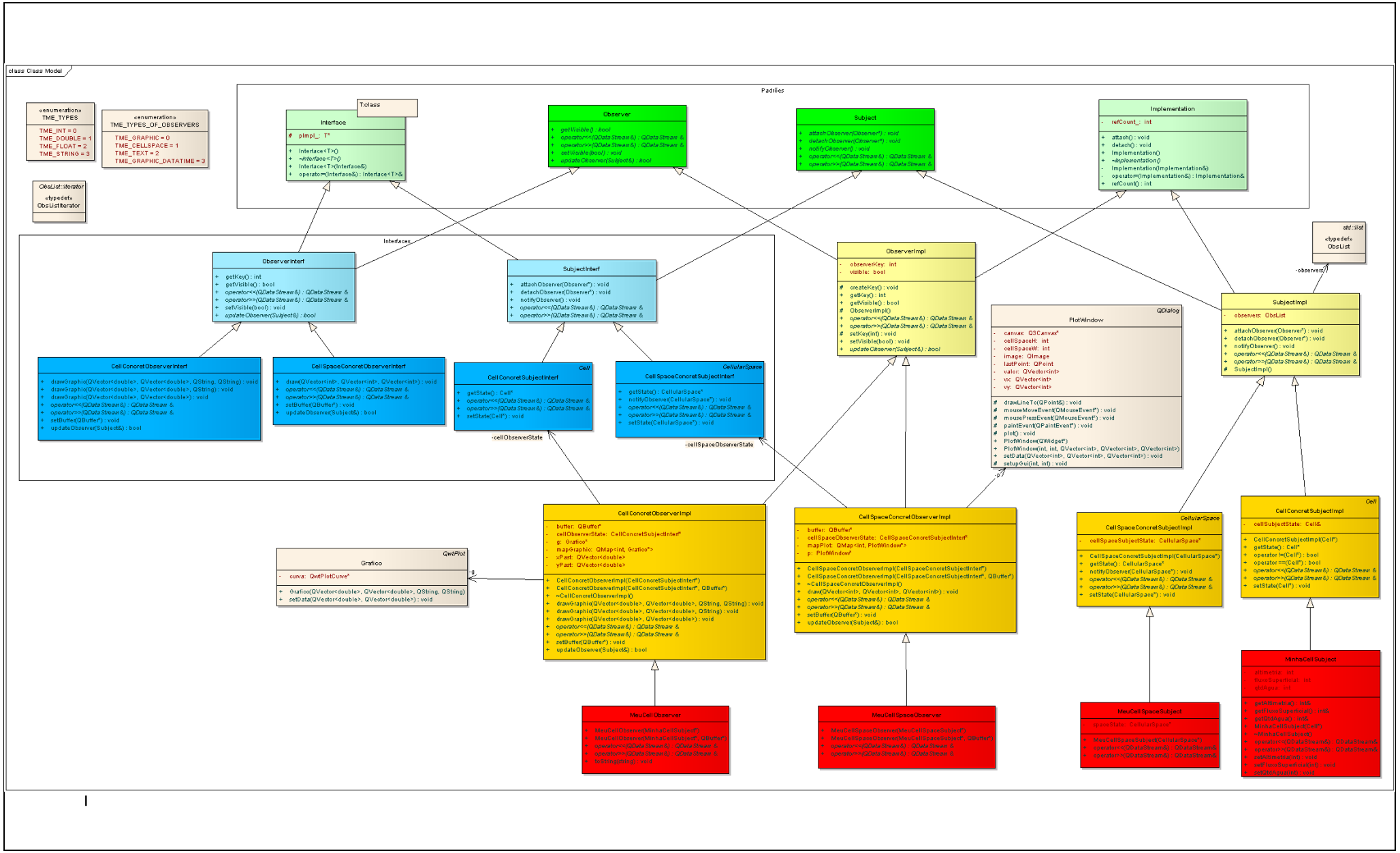

Figura 5 – Diagrama UML completo

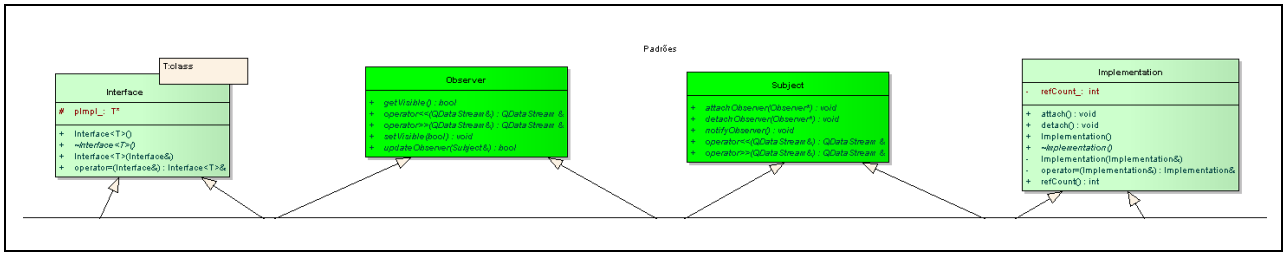

Figura 6 – Diagrama UML: Padrões de Projeto

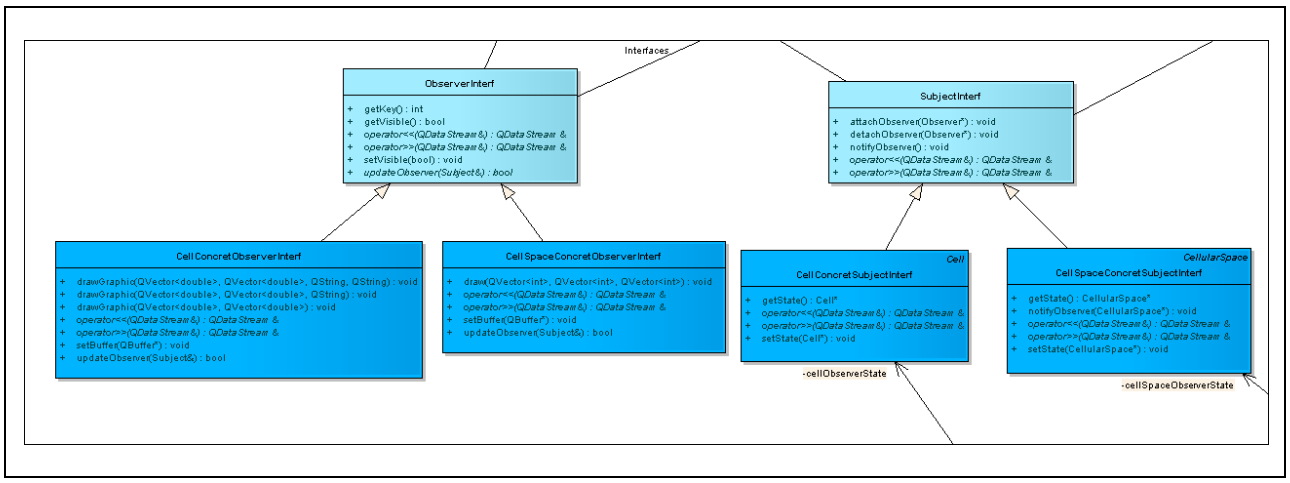

Figura 7 – Diagrama UML: Interfaces

```
(...)

MeuCellObserver* obs1 = (MeuCellObserver*) sub1->createObserver(TME_Graphic);

(...)
```

Exemplo 1 – Método de fabricação de observadores do tipo *TME_Graphic*

❖ As classes em tons de amarelo (Figura 8):

Essas são as implementações dos objetos, o código de execução de cada objeto esta contido nessas classes, as heranças permanecem idênticas àquelas das interfaces. Na segunda camada na hierarquia, em amarelo claro, apresentamos a implementação do padrão *Observer*. Em seguida, em amarelo escuro, temos as implementações dos objetos. As classes em marrom (Figura 5) são objetos responsáveis em gerar gráficos e a imagens dinâmicas. Logo abaixo, o Exemplo 2 e o Exemplo 3 ilustram a diferença

entre o código desenvolvido em uma interface e o código desenvolvido em uma implementação.

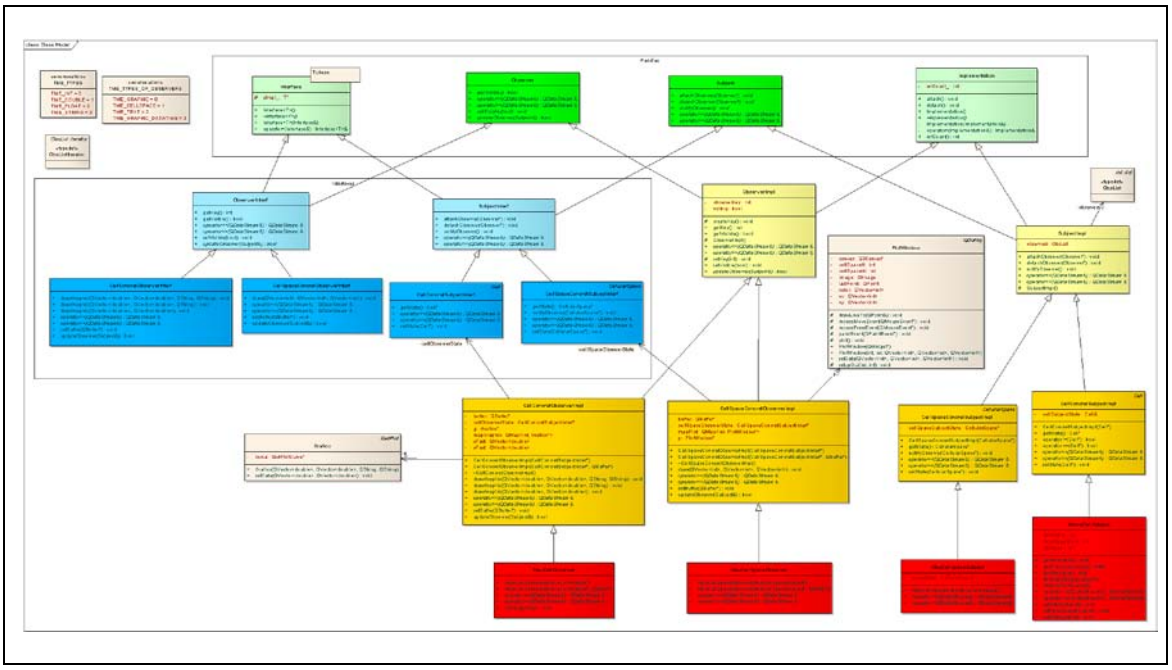

Figura 8 – Diagrama UML: Implementações

```
#include "CellConcretObserverInterf.h"
void CellConcretObserverInterf::setBuffer(QBuffer* b)
{
	((CellConcretObserverImpl*)pImpl_)->setBuffer(b);
}
```

Exemplo 2 – Código de uma interface, responsável por invocar o código de uma implementação

```
#include "CellConcretObserverImpl.h"
void CellConcretObserverImpl::setBuffer(QBuffer* b)
{
	QByteArray* a = new QByteArray( b->buffer());
	CellConcretObserverImpl::buffer = new QBuffer(a, 0);
}
```

Exemplo 3 – Código de uma implementação

6.4 – O problema de comunicação

O TerraME *Observer*, como já mencionado, irá proporcionar entre outras coisas a possibilidade de visualizar através de gráficos, tabelas, etc. os resultados do modelo em estudo. Entretanto, como produzir um observador de um modelo que existe apenas na "cabeça" do modelador e um *software* flexível, capaz de se adequar a qualquer modelo estudado? Em resposta a esta pergunta, desenvolvemos, baseado no *toolkit* QT, um mecanismo de serialização e desserialização, que irá prover a comunicação entre a camada do usuário final e o TerraME *Observer*. A Figura 9 demonstra essa comunicação. Desta forma, mesmo sem conhecer o modelo em estudo podemos criar observadores que permitem a visualização dos componentes do modelo. Assim, para utilizar o sistema, o modelador deverá implementar o mecanismo de serialização e desserialização em seu modelo de estudo, mantendo apenas o protocolo especificado. Abaixo, demonstramos o protocolo desenvolvido.

dados << "nº de parâmetro" << "tipos dos dados" << parâmetro1 << parâmetro2 <<...;

A seguir, o Exemplo 4, demonstra o método serializar em um objeto do tipo *subject* e o Exemplo 5, apresenta o método desserializar em um objeto do tipo *observer*.

```
// Implementação do método responsável por serializar(fragmentar) o objeto
QDataStream& MinhaCellSubject::operator<<(QDataStream& out)
{
    QString tipo;
    int qtdParametros = -1;
#ifdef DEBUG
    cout << "objeto cell serializado, classe MinhaCell" << endl;
#endif
    out << qtdParametros << tipo << quint32( altimetria ) << quint32( qtdAgua )
<< quint32( fluxoSuperficial );
    return out;
}
```

Exemplo 4 – Método de serialização em um objeto do tipo *subject*

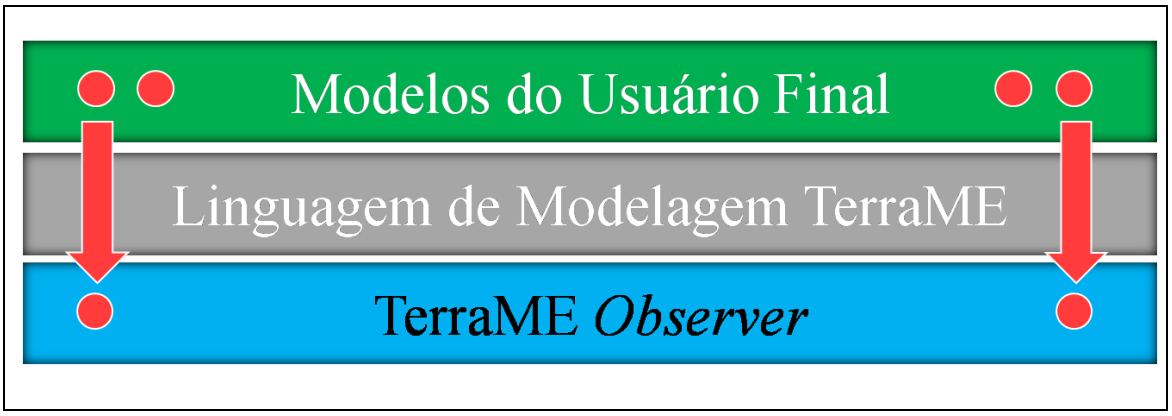

Figura 9 – Comunicação entre a camada do usuário final e o TerraME *Observer*

```
// Implementação do método responsável por desserializar (reagrupar) o objeto
QDataStream& MeuCellObserver::operator>>(QDataStream& in )
{
    int qtdParametros;
    QString tipo;
    int alt;
    int qAg;
    int fxSup;
#ifdef DEBUG
    cout << "objeto cell desserializado, classe MeuCellObserver" << endl;
#endif
    in >> qtdParametros >> tipo >> alt >> qAg >> fxSup;
    return in;
}
```

Exemplo 5 – Método de desserialização em um objeto do tipo *observer*

7 – Resultados

Alguns resultados já foram obtidos, por exemplo, observadores dinâmicos de células e de espaços celulares e o desenvolvimento de uma arquitetura concisa.

Para testar as funcionalidades desenvolvidas no TerraME *Observer*, criamos um simplificação do modelo desenvolvido na linguagem LUA chamado “Modelo de drenagem”, utilizando a linguagem C++. Tal modelo descreve o deslocamento de água ao descer uma encosta, em outras palavras, a chuva cai sobre as células de maior altitude que transfere a água para a célula vizinha mais baixa.

Nome	**Altimetria**	**Quantidade de água**	**Fluxo de água**
Célula 01	255	30	0
Célula 02	250	0	0
Célula 03	245	0	0
Célula 04	240	0	0

Tabela 1 – Valor dos atributos das células

Simplificamos o modelo como a seguir, o ambiente é representado como um espaço celular, onde todas as relações entre as células são descritas. Cada célula nesse espaço celular possui coordenadas, que as endereçam dentro do espaço. O ambiente foi reduzido para abranger apenas quatro células e cada uma, possuindo os atributos altimetria, quantidade de água, fluxo de água. Os valores desses atributos estão presentes na Tabela 1. A altimetria representa a altura dessa célula em relação ao nível do mar, a quantidade de água demonstra quanto de água essa célula possui e o fluxo representa o movimento de água entre células vizinhas.

Ao adicionarmos essas quatro células no espaço celular, as endereçamos conforme a Tabela 2:

Nome	Coordenada
Célula 01	(0, 0)
Célula 02	(0, 1)
Célula 03	(1, 0)
Célula 04	(1, 1)

Tabela 2 – Coordenadas no espaço celular

A chuva ocorre apenas na célula mais alta, altimetria igual a 255, então durante a simulação, são contados o número de vizinhos que possuem altimetria menor e em seguida, a quantidade de água é dividida igualmente entre eles e através do atributo fluxo, essa água é deslocada para os vizinhos. Esta ação ocorre para todas as células até que chegue naquela que possui menor altimetria que então fica alagada. Assim o fluxo retorna para a célula anterior que passará a ser inundada. A Figura 10 demonstra um observador de tipo gráfico que observa a quantidade de água da célula de menor altimetria, a cada instante de tempo, ela recebe mais água.

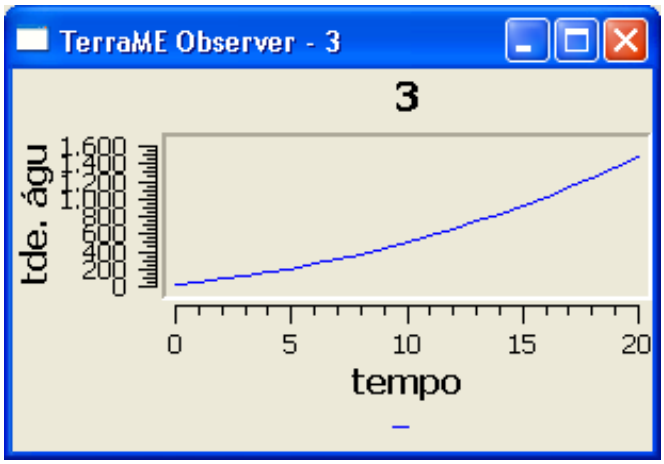

Figura 10 – Observador do tipo gráfico (quantidade de água X tempo)

A Figura 11 apresenta um observador de espaços celulares que demonstra a quantidade de água de todas as células inclusas naquele espaço. A célula em branco já está alagada e o fluxo de água está inundando a célula (em cinza) que também ficará totalmente encharcada tornando-a branca.

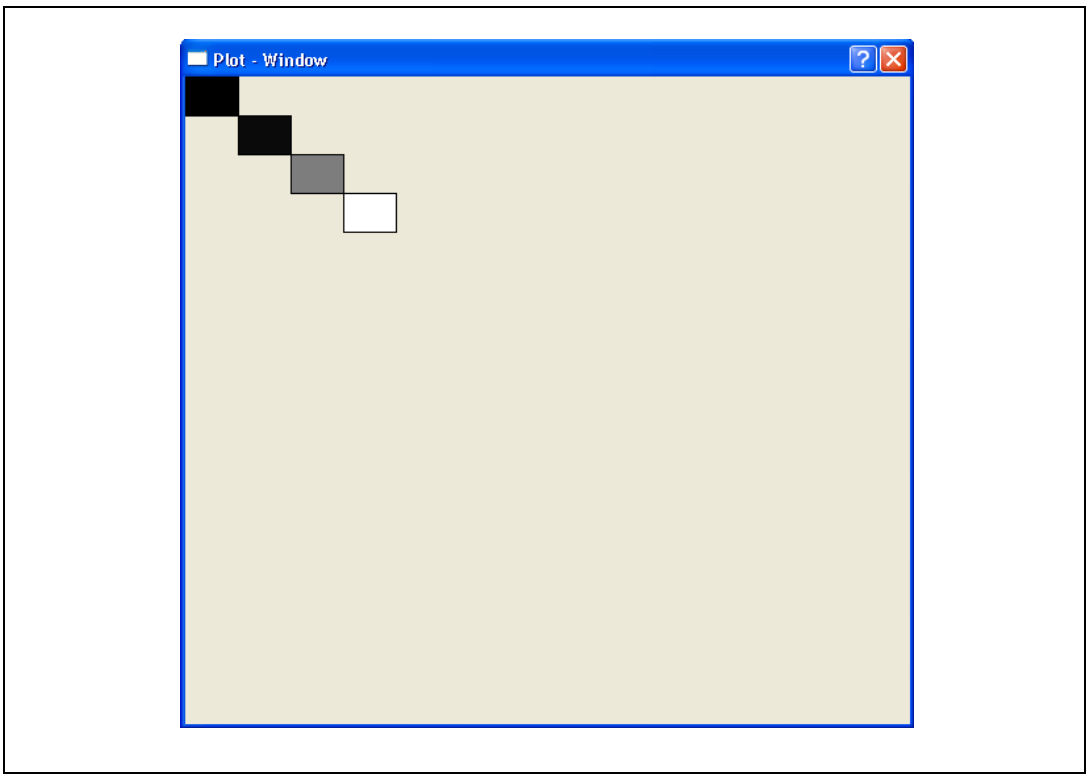

Figura 11 – Observador do tipo Espaço Celular

A Figura 12 apresenta toda a evolução deste modelo e a Figura 13 demonstra a quantidade de água da célula que possui menor altimetria.

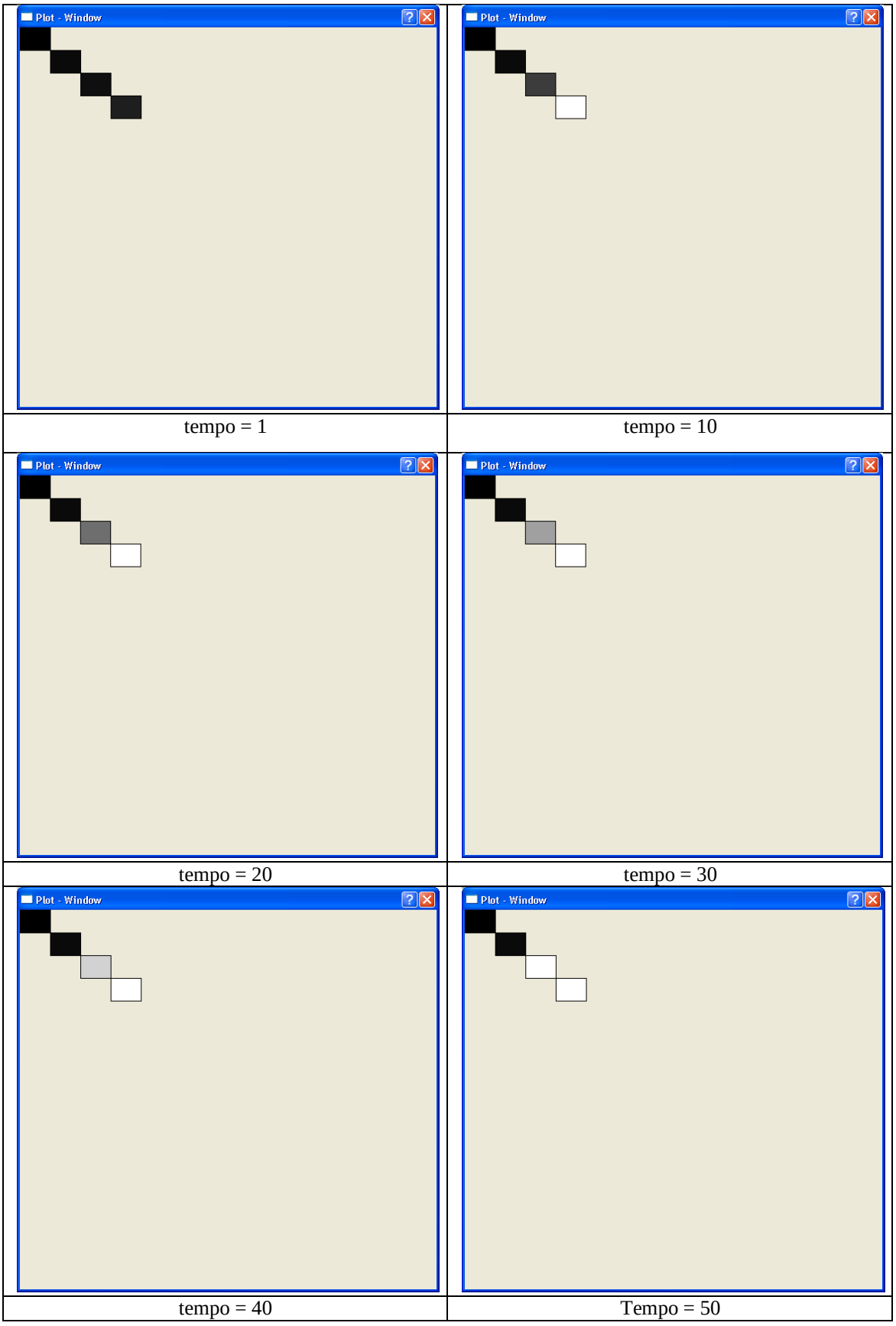

Figura 12 – Observador de espaço celular: evolução do modelo de drenagem

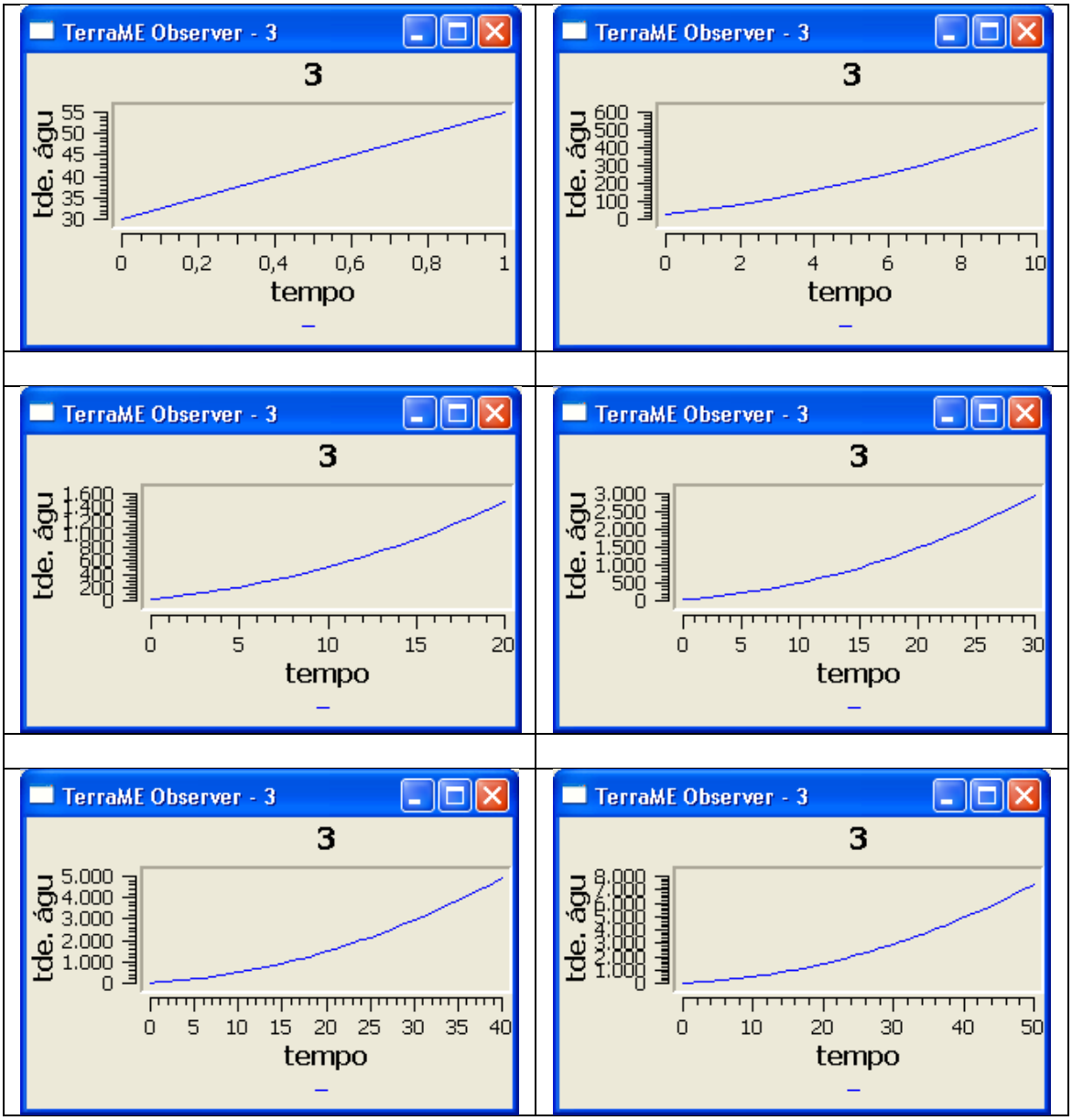

Figura 13 – Observador de célula: evolução do modelo de drenagem

8 – Conclusão

Uma simulação é usada em diversos ramos da ciência, ela proporciona formas de prever situações futuras. Assim, mecanismos de contingência e de mitigação podem ser produzidos de modo eficiente.

O ambiente de modelagem TerraME vem sendo utilizado em várias instituições para simular diversas situações, por exemplo, modelos ambientais e modelos epidemiológicos. Em todos os casos de estudo, ao término da simulação é necessário que análises sejam feitas para que medidas corretas possam ser tomadas.

Nesse trabalho, foi demonstrado a dificuldade do modelador em efetuar análises dos modelos de estudo e as diversas vantagens que o TerraME *Observer* trará, tais como, acompanhamento o*n-line* em gráficos, mapas e tabelas e a possibilidade de monitorar toda a dinâmica de uma simulação. Portanto, utilizando essa ferramenta, o tempo gasto atualmente desde a concepção do modelo até sua análise será drasticamente reduzido.

8.1 – Trabalhos futuros

No desenvolvimento desse sistema, alguns requisitos ainda não foram alcançados, mas estão pautados para os próximos ciclos de desenvolvimento. Alguns deles estão listados abaixo.

- Completa exportação do aplicativo para a linguagem LUA.
- Criação de mais observadores e a personalização deles.
- Geração da interface gráfica de modelagem.
- Mecanismos de depuração, avanço, retrocesso e pausa na simulação.

9 – Anexos

9.1 – Modelo de Drenagem: em linguagem LUA

```
--------------------------------------------------------------------------------
-- RAIN DRANAGE MODEL ( Author: Tiago Garcia de Senna Carneiro)
--------------------------------------------------------------------------------
-- Spatial dynamic modeling: the past is important
--------------------------------------------------------------------------------
-- CONSTANTS (MODEL PARAMETERS)
--------------------------------------------------------------------------------
C = 2; -- rain/t
K = 0.4; -- flow coeficient
FINAL_TIME = 10;
--------------------------------------------------------------------------------
-- GLOBAL VARIABLES
--------------------------------------------------------------------------------
csQ = CellularSpace{
dbType = "ADO",
host = "localhost",
database = " C:\\Documents and
Settings\\alunos\\Desktop\\CursoTerraME2007_Tiago\\cabecaDeBoi.mdb",
user = "",
password = "",
layer = "cellsLobo90x90",
theme = "cells",
select = { "altimetria", "qtdeAgua" }
}

--------------------------------------------------------------------------------
-- MODELO
--------------------------------------------------------------------------------
csQ:load();
CreateMooreNeighborhood(csQ);
ForEachCell( csQ, function( cell ) cell.capInf = 1; return true; end
);
csQ:synchronize();
for time = 1, FINAL_TIME, 1 do
-- PROCESSO: chuva
ForEachCell( csQ,
function( cell )
--if( cell.altimetria > 254 ) then
cell.qtdeAgua = cell.past.qtdeAgua + C;
--end
return true;
end
);
csQ:synchronize();

--------------------------------------------------------------------------------
-- relatório
--------------------------------------------------------------------------------
print("t: "..time );
csQ:save( time, "agua", {"qtdeAgua"} );

--------------------------------------------------------------------------------
-- PROCESSO: drenagem
--------------------------------------------------------------------------------

ForEachCell( csQ,
function( cell )
cell.qtdeAgua = cell.past.qtdeAgua - K*cell.past.qtdeAgua;
```

```
return true;
end
);
csQ:synchronize();

-------------------------------------------------------------------------------
-- PROCESSO: balanco hídrico
-------------------------------------------------------------------------------
for i, cell in pairs( csQ.cells ) do
-- conta vizinhos mais baixos
numViz = 0;
ForEachNeighbor(
cell,
0,
function(cell, neigh)
if(cell.altimetria > neigh.altimetria) then numViz = numViz + 1 end
return true;
end
);

-- envia agua para o vizinhos
if( numViz > 0 ) then
-- calcula escoamento superficial para cada vizinho
escoamento = cell.past.qtdeAgua/numViz;

- transborda
cell.qtdeAgua = 0;
ForEachNeighbor(
cell,
0,
function(cell, neigh)
if(cell.altimetria > neigh.altimetria) then
neigh.qtdeAgua = neigh.past.qtdeAgua + escoamento;
end
return true;
end
);
end
end
csQ:synchronize();
end
```

9.2 – Modelo simplificado de drenagem: em linguagem C++

```
Cell* celAux1 = new Cell();
Cell* celAux2 = new Cell();
Cell* celAux3 = new Cell();
Cell* celAux4 = new Cell();

//CellConcretSubject
MinhaCellSubject* sub1 = new MinhaCellSubject( celAux1 );
MinhaCellSubject* sub2 = new MinhaCellSubject( celAux2 );
MinhaCellSubject* sub3 = new MinhaCellSubject( celAux3 );
MinhaCellSubject* sub4 = new MinhaCellSubject( celAux4 );

QByteArray* byteArray = new QByteArray();
QBuffer* buffer = new QBuffer( byteArray );

//buffer.open(QIODevice::WriteOnly);
MeuCellObserver* obs1 = (MeuCellObserver*) sub1-
>createObserver(TME_Graphic);

//MeuCellObserver* obs1 = new MeuCellObserver(sub1);
obs1->setBuffer(buffer);

//MeuCellObserver* obs2 = new MeuCellObserver(sub2);
MeuCellObserver* obs2 = (MeuCellObserver*) sub2-
>createObserver(TME_Graphic);
obs2->setBuffer(buffer);

//MeuCellObserver* obs3 = new MeuCellObserver(sub3);
MeuCellObserver* obs3 = (MeuCellObserver*) sub3-
>createObserver(TME_Graphic);
obs3->setBuffer(buffer);

//MeuCellObserver* obs4 = new MeuCellObserver(sub4);
MeuCellObserver* obs4 = (MeuCellObserver*) sub4-
>createObserver(TME_Graphic);
obs4->setBuffer(buffer);

//-----------------------------------------------
int cont = 0;
CellIndex idx;
CellularSpace* cs = new CellularSpace();

//Cria o espaço celular observável
MeuCellSpaceSubject* cellSpaceSubj = new MeuCellSpaceSubject( cs );

//Método de criação de observer
MeuCellSpaceObserver* cellSpaceObs = (MeuCellSpaceObserver*)
cellSpaceSubj->
createObserver(TME_CellSpace);
cellSpaceObs->setBuffer(buffer);

//posicionamento da celula no espaço
idx.first = 0;
idx.second = 0;
cellSpaceSubj->add(idx, sub1);

idx.first = 1;
idx.second = 1;
cellSpaceSubj->add(idx, sub2);

idx.first = 2;
```

```
idx.second = 2;
cellSpaceSubj->add(idx, sub3);

idx.first = 3;
idx.second = 3;
cellSpaceSubj->add(idx, sub4);
// aloca o tamanho do objeto MinhaCellSubject ocupa na memória para o passado da Cell
int SIZEMEM = sizeof(MinhaCellSubject);
int CHUVA = 2;
int K = 0.4;
int TEMPO = 250;

// Sincroniza o espaço celular
cellSpaceSubj->synchronize(SIZEMEM);

sub1->setAltimetria(255);
sub1->setQtdAgua(30);
sub1->setFluxoSuperficial(0);
sub1->setState(sub1);

sub2->setAltimetria(250);
sub2->setQtdAgua(0);
sub2->setFluxoSuperficial(0);
sub2->setState(sub2);

sub3->setAltimetria(245);
sub3->setQtdAgua(0);
sub3->setFluxoSuperficial(0);
sub3->setState(sub3);

sub4->setAltimetria(240);
sub4->setQtdAgua(0);
sub4->setFluxoSuperficial(0);
sub4->setState(sub4);

cellSpaceSubj->setState(cellSpaceSubj);
cellSpaceSubj->synchronize(SIZEMEM);

while (cont <= TEMPO ) {
////////////////////////////////////////////////////////////
// chuva
pair<CellIndex, Cell*> indexCellPair;
CellularSpace::iterator itr = cellSpaceSubj->begin();
int aux = 0;

// procura celula com altimetria > 254
while (itr != cellSpaceSubj->end())
{
indexCellPair = *itr;
aux = ((MinhaCellSubject*) indexCellPair.second)->getAltimetria();
if (aux > 254) {
aux = ((MinhaCellSubject*)indexCellPair.second->getPast())-
>getQtdAgua();
aux = aux + CHUVA;
((MinhaCellSubject*) indexCellPair.second)->setQtdAgua(aux);
}

itr++;
}
cellSpaceSubj->synchronize(SIZEMEM);
```

```
////////////////////////////////////////////////////////////
// drenagem
itr = cellSpaceSubj->begin();
int auxAgua=0;
while (itr != cellSpaceSubj->end())
{
indexCellPair = *itr;
auxAgua = ((MinhaCellSubject*)indexCellPair.second->getPast())-
>getQtdAgua();
auxAgua = auxAgua - K * auxAgua;
((MinhaCellSubject*) indexCellPair.second)->setQtdAgua( auxAgua );
itr++;
}
cellSpaceSubj->synchronize(SIZEMEM);

////////////////////////////////////////////////////////////
// balanço hídrico
pair<CellIndex, Cell*> indexCellPairBalanco;
CellularSpace::iterator itrBalanco = cellSpaceSubj->begin();
while (itrBalanco != cellSpaceSubj->end()){
MinhaCellSubject* cell;
MinhaCellSubject* vizinho;
itr = itrBalanco;
indexCellPair = *itr;
cell = (MinhaCellSubject*)indexCellPair.second;
vizinho = (MinhaCellSubject*)indexCellPair.second;
int numViz = 0;
int escoamento = 0;

// conta os vizinhos mais baixos
while (itr != cellSpaceSubj->end())
{
indexCellPair = *itr;
vizinho = (MinhaCellSubject*)indexCellPair.second;
if ((cell != vizinho) && (cell->getAltimetria() >= vizinho-
>getAltimetria()))
numViz++;
itr++;
}

////////////////////////////////////////////////////////////
// transbordamento
itr = itrBalanco;
indexCellPair = *itr;

cell = (MinhaCellSubject*)indexCellPair.second;
if( numViz > 0 ){
escoamento = cell->getQtdAgua()/numViz;
while (itr != cellSpaceSubj->end())
{
indexCellPair = *itr;
vizinho = (MinhaCellSubject*)indexCellPair.second;
if ((cell != vizinho) && (cell->getAltimetria() >= vizinho-
>getAltimetria
())){
vizinho->setFluxoSuperficial(vizinho->getFluxoSuperficial() +
escoamento);
}
itr++;
}
}
```

```
////////////////////////////////////////////////////////
itr = cellSpaceSubj->begin();
indexCellPair = *itr;
int i = 0;
while (itr != cellSpaceSubj->end())
{
indexCellPair = *itr;
cell = (MinhaCellSubject*)indexCellPair.second;
cell->setQtdAgua(cell->getFluxoSuperficial());
buffer->open(QIODevice::WriteOnly);
QDataStream out1(buffer);

////////////////////////////////////////////////////////
// serialização
switch( i ) {
case 0: {
cout << "obs1: " << cell->getQtdAgua() << endl;
// out << qtdParametros << TME_TYPES << x << y;
out1 << 4 << TME_Int << cont << cell->getQtdAgua();
buffer->close();
obs1->setBuffer(buffer);
break;
}
case 1: {
cout << "obs2: " << cell->getQtdAgua() << endl;
out1 << 4 << TME_Int << cont << cell->getQtdAgua();
buffer->close();
obs2->setBuffer(buffer);
break;
}
case 2: {
cout << "obs3: " << cell->getQtdAgua() << endl;
out1 << 4 << TME_Int << cont << cell->getQtdAgua();
buffer->close();
obs3->setBuffer(buffer);
break;
}
default:{

cout << "obs4: " << cell->getQtdAgua() << endl;
out1 << 4 << TME_Int << cont << cell->getQtdAgua();
buffer->close();
obs4->setBuffer(buffer);
break;
}}
i++;
itr++;
}
itrBalanco++;
}
cellSpaceSubj->setState(cellSpaceSubj);
cellSpaceSubj->synchronize(SIZEMEM);

////////////////////////////////////////////////////////
byteArray->clear();
buffer->open(QIODevice::WriteOnly);
QDataStream outCellSpace(buffer);
```

10 – Referências Bibliográficas

BLANCHETTE, J.; SUMMERFIELD, M. C++ GUI Programming with Qt 4, 1a. Edição Prentice Hall PTR New Jersey - EUA, 2006.
_____. C++ GUI Programming with Qt 3, 1a. Edição Prentice Hall PTR New Jersey - EUA, 2004.

CARNEIRO, Tiago G. S. Nested-CA: um fundamento para a modelagem de uso e cobertura do solo em múltiplas escalas. Tese de doutorado em Computação Aplicada apresentada ao Instituto Nacional de Pesquisas Espaciais, São José dos Campos – SP, INPE, 2006.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Padrões de projeto: soluções reutilizáveis de software orientado a objetos, 1a. Edição, Bookman Porto Alegre, 1995.

IERUSALIMSCHY, R. Programming in LUA, lua.org, 2a. Edição, 2006.
_____. A linguagem Lua. Disponível em: http://www.inf.puc-rio.br/~roberto/talks/ufrj-2007.pdf, acessado em 03 de outubro 2008.

MINAR, N.; BURKHART, R. et al. The swarm simulation system: a toolkit for building multi-agent simulation. Santa Fe: SFI, SFI Working Paper 96-06-042, 1996.

PARKER, D. C.; BERGER, T.; MANSON, S. M. LUCC Report #6: Agent-Based Models of Land-Use and Land-Cover Change, 2002. (Shorter workshop proceedings) Disponível em: www.cipec.org/research/biocomplexity/ABM_Report6.pdf, acesso em: 15 de novembro de 2008.

PRESSMAN, R. S., Software Engineering: A Practitioner's Approach, Nova York:McGraw-Hill, 6ª Edição, 2005.

ROBERTS, N.; ANDERSON, D. et al. Introduction to computer simulation: a system dynamics modeling approach. Reading, MA: Addison-Wesley, 1983.

VELDKAMP, A.; FRESCO, L. O. CLUE: a conceptual model to study the Conversion of Land Use and its Effects. Ecological Modeling 85: 253-270, 1996.