Projeto VisualTerraME

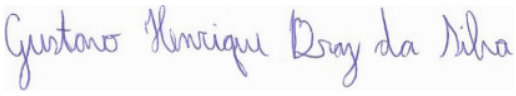
Relatório final de bolsa

Relatório apresentado à Fundação de Ciência, Aplicações e Tecnologia Espaciais – FUNCATE - relativo à concessão de bolsa de pesquisa na categoria Desenvolvimento Tecnológico e Industrial tipo III

Período: 01/10/2013 a 30/09/2014

Bolsista: Gustavo Henrique Braz da Silva

Assinatura do Bolsista:



Gustavo Henrique Braz da Silva

Assinatura do coordenador do projeto no INPE:

Dr. Antonio Miguel Vieira Monteiro

Setembro de 2014

Sumário

Introdução

A Modelagem Analítica e Simulação de Sistemas têm como objetivo entender fenômenos complexos, para que seja possível prever os seus acontecimentos, entender os motivos pelos quais eles ocorreram, simular tomadas de decisões e perceber no que essas tomadas de decisões podem acarretar. Neste contexto, simulações dos Sistemas Terrestres têm como objetivo o entendimento das interações entre a natureza e a sociedade e suas consequências, fornecendo base científica a definição de políticas publicas.

A principal dificuldade na modelagem e simulação computacional é que, na maioria dos casos, os especialistas no domínio do problema (biólogos, antropólogos, economistas, ecólogos, etc) nem sempre possuem conhecimento avançado em programação de computadores. Muitas vezes, isto dificulta representação e limita a complexidade do modelo do fenômeno em estudo. Para reduzir estes problemas, diversas as plataformas de modelagem e simulação têm evoluído para permitir o desenvolvimento de modelos por meio de interfaces gráficas. As plataformas NetLogo (Wilensky et. al, 2004) e o Repast (North et. al, 2007) se destacam em meio as demais, com expressiva comunidade de usuários na Europa e Estados Unidos, respectivamente. No entanto, apesar terem sido aplicadas em diversos domínios ambientais, elas não foram desenvolvidas especificamente para oferecer suporte a este nicho. Por isto, ignoram a necessidade do uso simultâneo de múltiplos paradigmas de modelagem (dinâmica de sistemas, multiagente e autômatos celulares) para representar as interações entre os sistemas naturais e sociais em múltiplas escalas temporais, espaciais e comportamentais (Carneiro et. al, 2013).

Nos últimos 8 anos, a Universidade Federal de Ouro Preto (UFOP) e o Instituto Nacional de Pesquisas Espaciais (INPE) vêm colaborando para desenvolver a plataforma TerraME para modelagem e simulação dos Sistemas Terrestres. Ele permite a representação de modelos multiparadigma e multiescala por meio de uma extensão da linguagem de programação Lua[1]. Contudo, não possui uma interface gráfica amigável que permita o desenvolvimento de modelos (Carneiro et. al, 2013).

Neste trabalho, foi implementado um ambiente de desenvolvimento integrado (IDE) baseado em programação visual, chamado Visual TerraME (VTME). Ele permite o desenvolvimento de modelos em TerraME a partir da composição de elementos da interface gráfica dos modelos. O VTME aborda o processo de modelagem de fora para dentro, isto é, o desenvolvimento acontece a partir daquilo que é mais próximo ao usuário do modelo (seus aspectos visuais) para aquilo que é mais distante do usuário do modelo (seus aspectos funcionais). A elementos de interface gráfica do VTME permitem a completa descrição da estrutura do modelo e parâmetros (parte descritiva do modelo) de forma visual, sem a necessidade de programação. O VTME deixa ao modelador a tarefa de programar somente aquilo que não se pode evitar, as regras do modelo (parte comportamental do modelo).

[1] http://www.lua.org/

Justificativa

O *TerraME* é utilizado pelo INPE (Instituto Nacional de Pesquisa Espacial) para responder questões relacionadas às Mudanças de Uso e Cobertura do Solo na região Amazônica (Câmara, 2007). Os pesquisadores da FIOCRUZ (Fundação Oswado Cruz) utilizam o *TerraME* para avaliar questões relativas ao Controle da Dengue (Lana, 2010) (Lana, 2011). As questões avaliadas pela EMBRAPA (Empresa Brasileira de Pesquisa Agropecuária), utilizando o *TerraME*, foram sobre os efeitos das erosões acelerados por mudanças climáticas sobre a produção de grãos (Martorano, 2009).

O *TerraME* está e constante uso, mas apesar disso, existe uma dificuldade dos usuários em desenvolverem novos modelos e evoluir aqueles que existem. Isto se deve ao fato de que, na maioria dos casos, os usuários potênciais do *TerraME* não possuem domínio de técnicas de programação. Porém, são eles que têm o conhecimento acerca dos fenômenos a serem modelados. Esta dificuldade em programação de computadores cria uma barreira, que impede os modeladores de progredirem no desenvolvimento de seus modelos.

Modelagem e Simulação de Sistemas: Revisão Bibliográfica

Nesta seção de texto, é apresentada uma revisão bibliográfica dos conceitos necessários para o entendimento deste trabalho, tais como modelagem e simulação, padrão de projeto, *factory*, *model view controller*, *TerraME*.

Modelagem e Simulação

A modelagem, de forma geral, pressupõe um processo de criação e descrição, envolvendo um determinado grau de abstração que, na maioria das vezes, acarreta numa série de simplificações dos fenômenos do mundo real. De tal forma que para lidar com tais fenômenos é necessária uma visão ampla e geral dos mesmos, considerando o conjunto de características que dê sustentação ao fenômeno em relação ao objeto em estudo (Hannon, 2001).

O aspecto mais importante de um modelo é a relação simplicidade versus fidelidade. Um modelo é a representação do conhecimento e a principal ferramenta para o estudo do comportamento de sistemas complexos. Modelar é o primeiro passo para a análise de um sistema de qualquer natureza e sob qualquer aspecto. Quando o modelo é uma representação válida de um sistema, informações significativas podem ser retiradas sobre sua dinâmica ou seu desempenho (Trivelato, 2003).

Processo de Modelagem

De acordo com Carneiro (Carneiro, 2006), o processo de modelagem de fenômenos espaciais dinâmicos, ocorre de forma iterativa e incremental, conforme ilustrado na Figura 1, e compreende as seguintes fases: (a) desenvolvimento da base de dados; (b) desenvolvimento do modelo; (c) calibração, verificação e validação do modelo; (d) execução e visualização do modelo e análise de relatórios; (e) projeção de cenários. As atividades de desenvolver modelos para sistemas ambientais e desenvolver software baseado no modelo espiral, apresentado por (Boehm, 88), são essencialmente similares, e envolvem as seguintes "macro-atividades": concepção, projeto, implementação e testes.

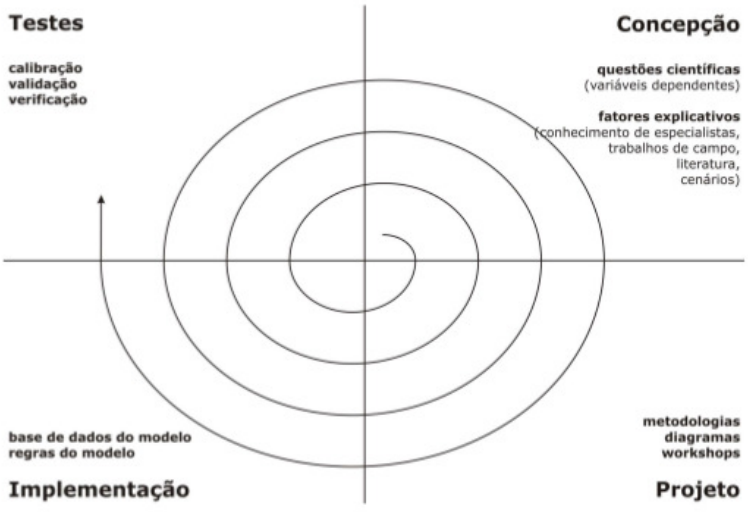


Figura 1: Processo iterativo e incremental de desenvolvimento de modelos. Fonte: Carneiro (2006)

Padrão de Projeto

Um padrão de projeto na computação descreve um problema e uma solução convencionada para resolver este problema, ele também define um nome para o dado padrão. A solução se torna um padrão, pois já foi utilizada, com sucesso, em outros projetos sofrendo apenas pequenas adaptações para cada situação\cite{gamma:1995}.

Definido por Erich Gamma como: "(...) são descrições de objetos e classes comunicantes que são customizados para resolver um problema geral de projeto num contexto particular". (gamma, 1995).

Factory

O padrão Factory define uma interface para a criação de objetos, a partir dessa interface são implementadas subclasses que definirão como um determinado objeto será instanciado (gamma:1995).

Model View Controller

Como definido por (Glenn, 1988) a arquitetura de software *Model View Controller* define a separação do *software* em três partes, de forma a separar cada parte em módulos visando à reutilização e independência da lógica e da visualização do sistema. A arquitetura MVC divide o sistema nos seguintes componentes:

- ***Model:*** São os dados, a lógica e as regras de negocio da aplicação.
- ***View:*** É a forma como o *Model* será visualizado, sendo possível que um mesmo dado do *Model* seja visualizado de formas diferentes.
- ***Controller:*** O *Controller* é responsável por gerenciar os eventos gerados pelos usuários, delegando cada ação para a *View* ou para o *Model*.

Plataformas de Modelagem e Simulação: Trabalhos Relacionados

Dentre os trabalhos presentes na literatura, o *NetLogo* e o *Repast* são as ferramentas das quais esse trabalho mais se aproxima, e são os mais relevantes, servindo de base para outros trabalhos.

NetLogo

Como descrito por (Tisue, 2004) o *NetLogo* é uma linguagem de programação *multi-agent* e um ambiente de modelagem e simulação para fenômenos naturais e sociais. Ele é uma aplicação *standalone* desenvolvida em Java, capaz de executar na maioria das plataformas computacionais, voltado para a pesquisa e a educação. O *NetLogo* é bastante flexível e extensível, capaz de ser estendido para qualquer linguagem que utilize a JVM (*Java Virtual Machine*), como por exemplo, Java e Scala[2]. O *NetLogo* gera a maioria do código necessário para a modelagem facilitando, o desenvolvimento do modelo, no entanto, ele não oferece facilidades gráficas ao usuário quando o mesmo está definindo a parte lógica e aritmética do seu modelo, fazendo com que ele tenha que especifica-la inteiramente utilizando a linguagem definida pelo *NetLogo*.

Repast Simphony (Repast S)

Segundo (North, 2005) *Repast Symphony* é uma extensão do *Repast* (Nnorth, 2007) que oferece uma abordagem de simulação, desenvolvimento e execução, voltada para a especificação comportamental de agentes e de modelagem auto-construtiva. O *Repast Symphony* pode ser utilizado para desenvolver de diferentes formas, incluindo *point-and-click statecharts*, Groovy[3], ou Java, ou utilizando todos eles, de forma intercalada. Apesar de ser bastante flexível e apresentar várias formas de desenvolvimento, o *Repast Symphony* ainda necessita que o usuário tenha um conhecimento básico sobre o paradigma de programação POO (Programação Orientada a Objetos), o que dificulta sua utilização para modeladores, sem conhecimento desse paradigma.

[2] http://www.scala-lang.org/
[3] http://groovy.codehaus.org/

Plataforma de Modelagem *TerraME*

Este capítulo apresenta a plataforma *TerraME*, um ambiente de modelagem e simulação de sistemas espaciais dinâmicos que oferece estruturas de dados e serviços para a construção e simulação de modelos ambientais, através da linguagem de programação *TerraME*.

Plataforma TerraME

O *TerraME* (*TerraLib Modelling Environment*) (Carneiro, 2006) é destinado à implementação e simulação de modelos ambientais que envolvam a representação explícita do espaço. Desenvolvido a partir do trabalho de doutorado de Carneiro (Carneiro, 2006), no Instituto Nacional de Pesquisas Espaciais (INPE), concluído em 2006. Atualmente é mantido pelo TerraLAB (Laboratório para Modelagem e Simulação de Sistemas Terrestres), uma parceria entre o INPE e a Universidade Federal de Ouro Preto (UFOP).

O *TerraME* permite a representação e simulação de modelos espaciais dinâmicos integrados a um sistema de informações geográficas. Os componentes de sua arquitetura de software oferecem serviços específicos a usuários com diferentes níveis de conhecimento em algoritmos e técnicas de programação. Usuários experientes podem implementar modelos utilizando diretamente o framework de modelagem *TerraME* através da linguagem de programação C++, enquanto aqueles que possuem apenas o conhecimento básico sobre algoritmos e modelagem computacional podem utilizar a linguagem de programação de alto nível *TerraME Modeling Language* - uma extensão da linguagem de programação Lua (Ierusalimschy, 1996), que permite a fácil escrita, leitura e alteração dos modelos (Carneiro, 2006).

Arquitetura do TerraME

O *TerraME* foi construído baseado na arquitetura em camadas, onde as camadas inferiores fornecem funcionalidades sobre as quais as camadas superiores são implementadas, conforme mostrado na Figura 2.

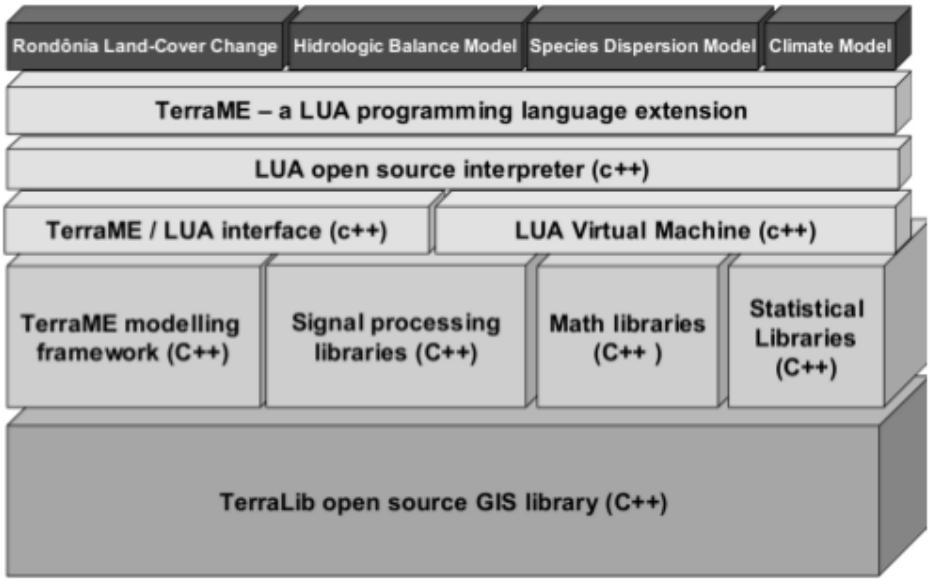


Figura 2: Arquitetura da plataforma *TerraME*. Fonte: Carneiro (2006)

Conforme definido por Carneiro (2006), na camada inferior, a biblioteca TerraLib fornece serviços para gerenciamento e análise de dados espaço-temporais. Na segunda camada, o *TerraME Modeling Framework* fornece serviços para a simulação, calibração e validação de modelos que podem ser utilizados através da linguagem C++. Nesta camada, o modelador tem acesso a um poderoso conjunto de serviços para modelagem e simulação. Contudo, sua interface de programação apresenta uma sintaxe complexa e, por isso, de difícil utilização para usuários que não possuem sólidos conhecimentos em técnicas de programação e na linguagem C++. A camada seguinte é aquela formada pelo interpretador e pelo ambiente de execução da linguagem de modelagem *TerraME*, que estende a linguagem de programação LUA pela inclusão de novos tipos de dados, especialmente projetados para a modelagem espacial dinâmica, e de serviços para simulação e avaliação de modelos. Sobre as demais, está a camada de aplicação, composta por modelos ambientais desenvolvidos pelos usuários da arquitetura. Desta forma, a construção de modelos na plataforma pode ser feita a partir das linguagens de programação C++ e *TerraME*.

Linguagem *TerraME*

A linguagem de programação *TerraME Modeling Language*, uma extensão da linguagem LUA, permite a representação de modelos espaciais dinâmicos a partir dos seguintes tipos de dados:

- ***Environment:*** utilizado para representar o conceito de ambiente (escala) e permitir o desenvolvimento de modelos que considerem múltiplas escalas;
- ***CellularSpace, Cell, Neighborhood:*** utilizados para representar o espaço, suas propriedades e relações topológicas;
- ***Agent, Automaton, State, Jump, Flow e Trajectory:*** utilizados para representar o comportamento de sistemas - processos ou atores que dinamicamente alteram as propriedades do espaço e interagem entre si;
- ***Timer, Event e Message:*** utilizados para representar o tempo e definir o momento e a ordem na qual os eventos serão executados.

Desenvolvimento do *VisualTerraME*

Neste capítulo é apresentada a estratégia utilizada para a implementação do Ambiente de Desenvolvimento Integrado (IDE), denominado *VisualTerraME*, o processo de desenvolvimento, a arquitetura do software e a interface gráfica implementada.

Processo de Desenvolvimento

O *VisualTerraME* foi desenvolvido aplicando o processo de desenvolvimento BOPE (Muzetti, 2014), a divisão da estrutura de desenvolvimento está descrito da seguinte maneira:

- ***Backlog do projeto:*** Nesta etapa são definidas as questões que a ferramenta *VisualTerraME* deve responder. Os trabalhos relacionados são investigados. As informações disponíveis são analisadas na esperança de que sejam identificados todos os requisitos de *software* e da arquitetura;

- ***Estórias de Usuários:*** Nesta etapa as principais necessidades dos usuários, ou de potenciais usuários, são levantadas em forma de relatos, expectativas ou estórias vividas pelos usuários, de forma a entender como o *VisualTerraME* deve se comportar em uma dada situação (estória de usuário), essas estórias servem como direcionador de como deve ser a implementação e os testes que validam a ferramenta;

- ***Cronograma do Projeto:*** Nesta etapa é definida a prioridade em que as estórias do usuário devem ser implementadas, aqui também é quando se define as datas, o que deve ser feito em cada *sprint*, mesmo que, por algum motivo, seja necessário modificar o cronograma é importante defini-lo completamente para fazer uma primeira estimativa e, ao longo do desenvolvimento, refinar as estimativas;

- ***Sprint:*** Uma *sprint* é marcada por uma reunião inicial, aonde se planeja quais requisitos do *Backlog* devem ser desenvolvidos para completar uma dada estória do usuário. As *sprints* são definidas como um conjunto de iterações, e ao final de cada *sprint* espera-se uma nova versão do *VisualTerraME*, com as estórias selecionadas no inicio implementadas e validadas;

- ***Iteração:*** Uma iteração tem a duração de 5 dias, sendo previstos dentro desse intervalo, uma reunião de planejamento, para decidir quais os requisitos da *sprint* serão implementados e uma reunião de validação, para apresentar as implementações, já testadas, daquela iteração.

Ferramenta Proposta

O *VisualTerraME* foi concebido para facilitar a criação e simulação de modelos, para modeladores que não têm domínio de técnicas de programação. A plataforma permite o desenvolvimento de modelos *TerraME* por meio de elementos gráficas, que remetem aos elementos presentes no *TerraME*, de forma a facilitar e agilizar o desenvolvimento.

O *VisualTerraME* também executa o modelo, utilizando o *TerraME* internamente, permitindo que o usuário intervenha nas propriedades do modelo enquanto o modelo está em execução.

O *VisualTerraME* foi desenvolvido em C++, utilizando o *framework Qt Creator*

e seguindo os padrões de projetos apropriados.

Componentes de Interação

O *VisualTerraME* é definido em quatro componentes de interação, sendo eles *Model Structure*, *Property Editor*, *Console* e *Form*.

Model Structure

O *Model Structure* organiza todos os *subjects* de forma hierárquica como pode ser observado na Figura 3, possibilitando que cada *subject* crie um *subject* filho, o que define a composição de sub modelos presente no *TerraME*. Esta organização facilita o usuário a entender como os *subjects* estão se relacionando.

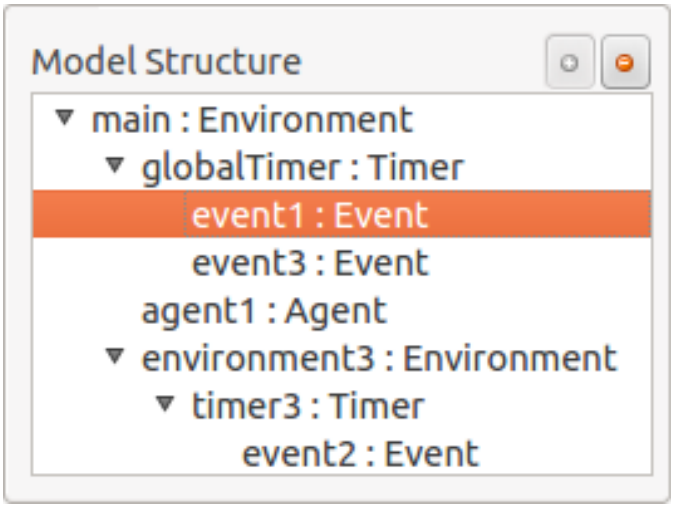


Figura 3: Exemplo de visualização hierárquica do *Model Structure*

Form

O *Form* é a área onde os objetos com representação gráfica são visualizados, seja esses objetos, inputs ou *observers*, esses objetos podem ser colocados na posição preferencial do usuário, o que facilita a visualização e entendimento do modelo, como pode ser visto na Figura 4.

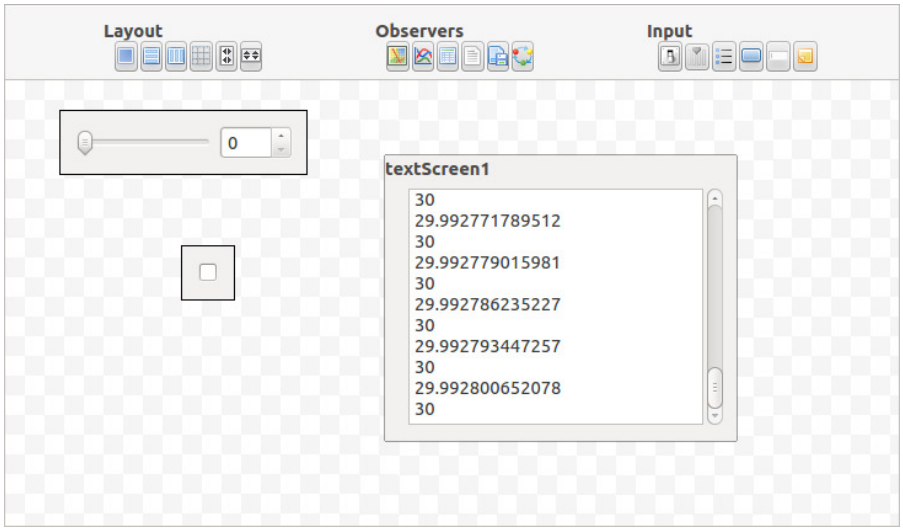


Figura 4: Exemplo de visualização da área dos elementos gráficos.

Property Editor

Definido com uma tabela de propriedades, o *Property Editor* é onde o usuário define quais serão as propriedades do elemento selecionado, este elemento pode fazer parte do *Model Structure* (*Subject*), ou ser um objeto gráfico presente no *Form*. A Figura 5, demonstra um exemplo de *Property Editor*.

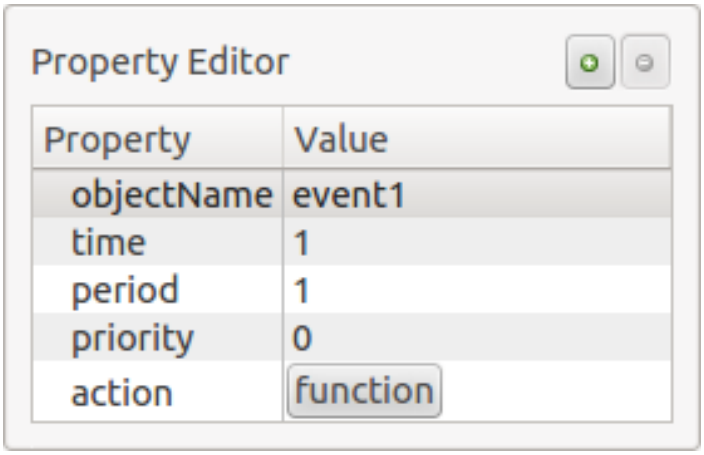


Figura 5: Exemplo do *Property Editor* vinculado ao *Subject Event*

Console

O *Console* é utilizado para mostrar a saída padrão (*stdout*) do modelo, como pode ser observado na Figura 6. Tudo que é enviado para a saída padrão como resultado da execução do modelo *TerraME* será exibido no *Console*.

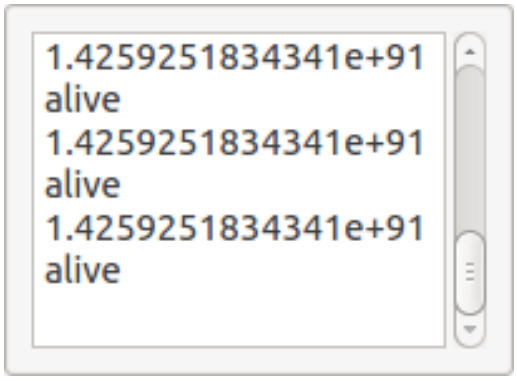


Figura 6: Exemplo de *Console* com uma saída ilustrativa

Input

Um *Input* é como o usuário pode interagir em algum *Subject* enquanto o modelo está sendo simulado. Um *input* pode conter uma ou mais propriedades de um ou mais *Subjects* vinculados, com isso o valor dessas propriedades será modificado quando o *Input* for alterado.

Os *Inputs* presentes no *VisualTerraME* são:

- ***Slider:*** Utilizado para modificar um valor numérico definido dentro de um intervalo definido em sua propriedade;
- ***Switch:*** Utilizado para modificar valores *booleanos* (*true, false*);
- ***Button:*** Utilizado para ativar uma função pré-definida pelo modelador, quando

pressionado;

- ***List:*** Utilizado como uma lista de valores discretos enumerados. O *List* modifica o valor de alguma propriedade vinculada à ele;
- ***Note:*** Utilizado para criar campos textuais, para inserir observações referentes ao modelo.

Observer

Um *Observer* do *VisualTerraME* é uma extensão do *TerraME Observer* definido em (Rodriguez, 2012), ele é utilizado para visualizar propriedades dos *Subjects* vinculadas à ele em tempo de execução. Os *observers* que estão presentes no *VisualTerraME* são:

- ***Map:*** Uma representação do *CellularSpace* utilizado para visualizar uma ou mais propriedades dos *Agent, Society* ou *Trajectory* associados a um *CellularSpace*;
- ***Chart:*** Um gráfico dinâmico utilizado para visualizar o comportamento de uma ou duas propriedades numéricas durante a simulação;
- ***Table:*** Uma tabela que é utilizada para visualizar o comportamento das propriedades de um *subject* durante a simulação;
- ***TextScreen:*** Um texto que a cada nova linha mostra os valores de cada atributo atualizados na iteração;
- ***StateMachine:*** Uma máquina de estado utilizado para visualizar as transições de estados de um *Automaton* ou *Agent*.

Estratégias de Desenvolvimento

Utilizando os conceitos da Programação Orientada a Objetos (POO), algumas estratégias e padrões de projetos foram implementados com o objetivo de solucionar determinados problemas.

Factory

O padrão de projeto *Factory* foi implementado para a criação de objetos complexos, que necessitam de um gerenciamento mais especifico sobre sua criação ou remoção. São eles os *Subjects* e Elementos Gráficos, em ambos os casos, sua remoção ou inserção interfere em todo o sistema, sendo para renderizar o *Model Structure, Property Editor* ou *Form*, ou para remover uma hierarquia inteira do *Model Struct*ure e remover todos os vínculos com os *Inputs* ou *Observers*.

Render Delegation

A estratégia *Render Delegation* (Delegação da Renderização) faz uso do polimorfismo e sobrecarga de métodos presentes na POO, de forma a definir para cada objeto como ele será renderizado, com isso, ao invés do *Viewer* renderizar os objetos, os objetos utilizam o *Viewer* para se renderizar, essa estrategia permite que cada *Object* possa definir como será feita a renderização de sua *Property Editor*, no caso dos *Subjects*, também define como eles serão exibidos no *Model Structure*. Essa estratégia é utilizada também para definir como cada objeto é carregado, ou o que ele precisa salvar.

Subject

Um *Subject*, no *VisualTerraME*, não possui representações gráficas que podem remeter

a símbolos do mundo real, por isso um *Subject* não pode ser visualizado em um *Form*, para visualizar um *subject* deve-se utilizar um *Observer* apropriado.

Elemento Gráfico

Um elemento gráfico do modelo é formado por duas classes, como pode ser visto no exemplo do *Slider* na Figura 7. A classe *Slider*, que é uma subclasse de *Input* representa a parte lógica do objeto, onde os valores e os vínculos do *slider* estão armazenados.

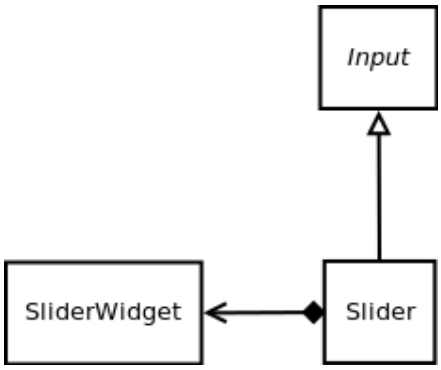


Figura 7: Relacionamento de Classe entre o *Input, Slider* e *SliderWidget*

A classe *SliderWidget* é a representação gráfica do *Slider* e define como o *slider* é desenhado e todos os seus eventos. Esses objetos definem como o elemento gráfico será criado pelo *factory GraphicsWidget*, Figura 8, que faz o vinculo entre o *Slider* e *SliderWidget* e retorna um objeto *QGraphicsItem*[4] que será visualizado no *Form*.

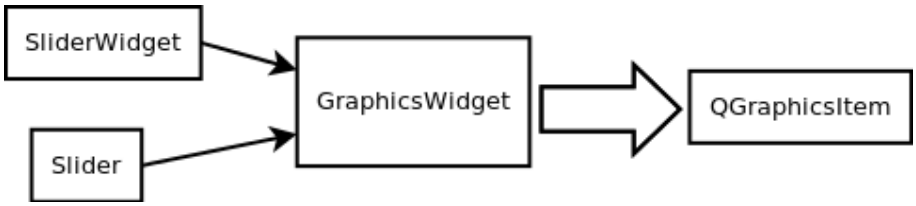


Figura 8: Exemplo da criação do *Slider* pelo *factory GraphicsWidget*

Arquitetura

O *VisualTerraME* é definido utilizando a arquitetura MVC (*Model View Controller*), como poder ser observado na Figura 9, em que *Model* representa todos os elementos que compõe o modelo. O *Controller* é responsável por modificar os valores do *Model*, e notificar (*notify*) o *Viewer* que algum elemento do modelo foi modificado e que será necessário que ele seja renderizado novamente. Ele gerencia os eventos vindos das interações com o usuário por meio da *Viewer*. O *Controller* também é responsável por enviar, ao *Viewer*, os elementos que ele requisitar.

[4] http://qt-project.org/doc/qt-4.8/qgraphicsitem.html

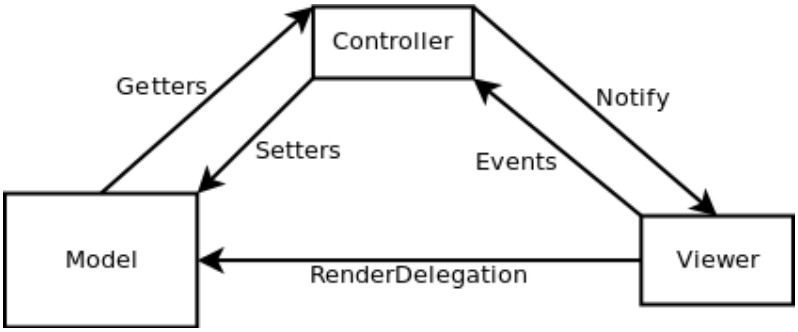


Figura 9: Arquitetura MVC utilizada no *VisualTerraME*

Fluxo de Execução

A execução do modelo pelo *VisualTerraME* é descrita pela Figura 10. O Ambiente de Desenvolvimento é utilizado para facilitar o desenvolvimento do modelo pelo usuário. Após o desenvolvimento, o modelo *VisualTerraME* é traduzido para um código *TerraME*, fazendo o mapeamento de cada objeto do modelo em objetos do *TerraME*. Depois, o *TerraME* é executado, e todas as saídas são enviadas ao Ambiente de Desenvolvimento para serem exibidas ao usuário. O usuário pode, a partir do Ambiente de Desenvolvimento, interagir com a simulação em execução utilizando os *Inputs*.

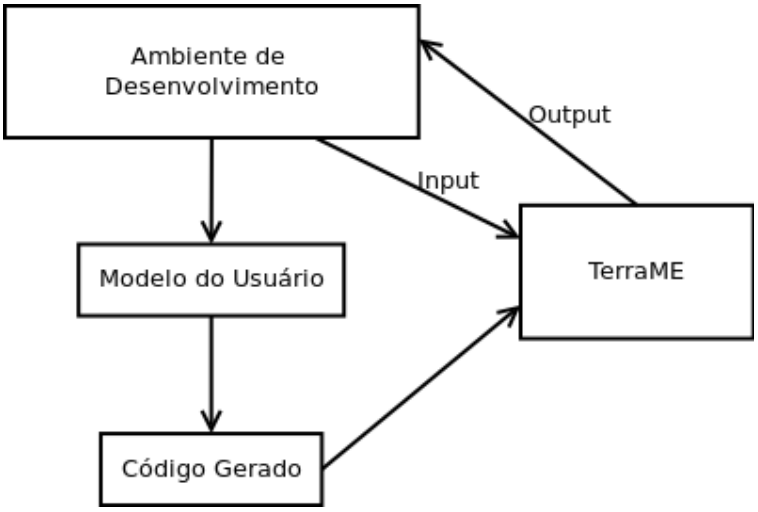


Figura 10: Modelo do fluxo de execução do *VisualTerraME*

Resultados Obtidos

Como resultado deste trabalho foi desenvolvido o *software VisualTerraME*, juntamente com os seguintes artefatos:

Manual de uso

Manual incentivando as boas práticas de utilização do *VisualTerraME*, visando ajudar os novos usuários a entenderem o funcionamento da ferramenta, quais as funcionalidades presentes e como utilizar da melhor maneira.

Distribuição multiplataforma

Assim como o *TerraME*, o *VisualTerraME* tem distribuição para 2 sistemas operacionais diferentes, *Windows* e *Linux*.

Bibliografia

Andrade, Antônio José C. Rodrigues and Tiago G. S. Carneiro and Pedro R. “TerraME Observer: An extensible real-time visualization pipeline for dynamic spatial models.” 2012.

B., Câmara G. and Escada I. and Aguiar A. P. and Amaral S. and Carneiro T. G. S. and Monteiro A. M. V. and Araujo R. and Vieira I. and Becker. “Amazonian Deforestation Models.” 2007.

BOEHM, B. W. “A Spiral Model of Software Development and Enhancement.” 61-72. Computer, 1988.

CARNEIRO, T. G. S. “Nested-CA: a foundation for multiscale modeling of land use and land change.” São José dos Campos, Tese (Doutorado em Computação Aplicada): Instituto Nacional de Pesquisas Espaciais (INPE), 2006.

Carneiro, Tiago Garcia Senna and Lima, Tiago França and Faria, Sérgio Donizette. “TerraLAB - Using Free Software for Earth System Research and Free Software Development.” *WSL 2009 - Workshop de Software Livre*, 2009.

Escada, A.P. Aguiar and G. Câmara and M.I.S. “Spatial statistical analysis of land-use determinants in the Brazilian Amazon: exploring intra-regional heterogeneity.” *Ecological Modelling*, 2007: 169-188.

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

Gattass, Gilberto Câmara and Ricardo Cartaxo and Modesto De Souza and Bianca Maria Pedrosa and Lúbia Vinhas and Antônio Miguel and Vieira Monteiro and João Argemiro Paiva and Marcelo Tilio and De Carvalho and Marcelo. “TerraLib: Technology in Support of GIS Innovation.” *in II Workshop Brasileiro de Geoinformática, GeoInfo2000. 2000. São Paulo. Case A: C07L04 neighbors ANNEX*, 2000.

HANNON, B., e M. RUTH. *Dynamic Modeling.* Springer-Verlag, New York, 2001.

IERUSALIMSCHY, R., L. H. FIGUEIREDO, e W CELES. “Lua - an extensible extension language.” 635-652. Software: Practice and Experience, 1996.

Lana, R. M. and Carneiro, T. G. S. and Honório, N. A. and Codeço, C. T. “Multiscale analysis and modeling of Aedes aegypti population spatial dynamics.” *Journal of Information and Data Management*, 2011: 211 --220.

Lana, R. M. and Carneiro, T. G. S. and Honório, Nildimar A. and Codeco, C. T. “Change Allocation in Spatially-Explicit Models for Aedes aegypti Population Dynamics.” *XI Simpósio Brasileiro Geoinformática*, 2010.

Martorano, L. and Meirelles, M. and Schuler, A. “Erosive Potential of Rains in the Climate Change Scenarios in the Upper Taquari River Basin, Brazil.” *Tropentag - Conference on Tropical and Subtropical Agricultural and Natural Resource Management*, oct de 2009: 6-8.

Ozik, J. and Collier, N.T. and Murphy, J.T. and North, M.J. “The ReLogo agent-based modeling language.” In: *Simulation Conference (WSC), 2013 Winter*, 1560-1568. 2013.

OZIK, M.J. NORTH and ERIC TATARA and N.T COLLIER and J. “VISUAL AGENT-BASED MODEL DEVELOPMENT WITH REPAST SIMPHONY.” 2007.

Pereira, Igor Muzetti. “Desenvolvendo software inovador em universidades públicas: Adaptando processos ágeis para a realidade de laboratórios de pesquisa e desenvolvimento.” *Universidade Federal de Ouro Preto (UFOP).* Master Thesis, mar de 2014.

Pope, Glenn E. Krasner and Stephen T. “A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System.” 1988.

T.G.S., Moreira E. and Costa S. and Aguiar A. and Câmara G. and Carneiro. “Dynamical coupling of multiscale land change models.” *Landscape Ecology*, 2009: 183-194.

Trivelato, Gilberto da Cunha. “Técnicas de modelagem e simulação de sistemas dinâmicos.” INPE - São José dos Campos, 2003.

VOS, M.J. NORTH and T.R. HOWE and N.T. COLLIER and J.R. “THE REPAST SIMPHONY DEVELOPMENT ENVIRONMENT.” *Generative Social Processes, Models, and Mechanisms.*, 2005.

Wilensky, Seth Tisue and Uri. “NetLogo: A Simple Environment for Modeling Complexity.” *Presented at the International Conference on Complex Systems*, may de 2004.

—. “NetLogo: Design and Implementation of a Multi-Agent Modeling Environment.” *SwarmFest, Ann Arbor*, may de 2004: 9-11.

Woods, R. C. Gonzalez and R. E. “Digital Image Processing.” 954. Prentice Hall, 2008.