Ferramentas de monitoramento e análise de modelos espaciais dinâmicos em tempo-real

Antônio José da Cunha Rodrigues, Tiago Garcia de Senna Carneiro
PPGCC - Programa de Pós-Graduação em Ciência da Computação
UFOP - Universidade Federal de Ouro Preto
Ouro Preto, Minas Gerais, Brasil
email: aj.rodrigues@ymail.com, tiagogsc@gmail.com

***Resumo*—A Visualização Científica se refere a qualquer técnica que utiliza visualização e dados científicos. É aplicada em diversas áreas pois sintetiza várias informações. Neste trabalho, usamos a visualização no monitoramento de variáveis de estado de modelo espaciais dinâmicos. Utilizando padrões de projeto, buscou-se obter desempenho no monitoramento de modelos através da reutilizar os dados recuperados do modelo. O uso dessa nova estrutura permitiu um ganho de desempenho da simulação a custo de um maior consumo de memória.**

***Keywords*-visualização científica, padrões de projeto, estrutura de dados, protocolo de comunicação, algoritmos, complexidade.**

I. Introdução

A Visualização Científica se refere a qualquer técnica que utiliza visualização e dados científicos. Segundo DeFanti *et al.* [1], é uma forma de comunicação que transcende a aplicação e os limites tecnológicos. As técnicas de visualização sintetizam várias informações de maneira eficaz [2] e são usadas em diversas áreas, por exemplo, Medicina, Engenharia e Ciência. Na Medicina, a visualização científica é aplicada em diversos casos, por exemplo, ressonância magnética e tomografia. Na Engenharia, também é usada em várias situações, como, na industria automobilística e em ferramentas *Computer-aided design* (CAD). Na Ciência, é aplicada em inúmeros casos, por exemplo, estudos da dinâmica populacional do mosquito da dengue e em estudos de fenômenos geográficos. Essa última aplicação é investigada neste trabalho. Para isso, aplicou-se a Visualização Científica para monitorar a evolução de simulações espacialmente-explícitas de processos naturais e sociais, ou simplesmente ambientais.

A modelagem computacional permite realizar experimentos controlados que aplicados ao estudo de fenômenos geográficos simulam os impactos ambientais provocados pelo homem que, por sua vez, condicionarão suas ações futuras. Tais simulações, geram uma enorme quantidade de dados que necessitam ser analisados de maneira eficiente. Antes da execução dos experimentos, os modelos devem ser verificados e quaisquer falhas detectadas devem ser corrigidas. O cérebro humano tem maior facilidade em reconhecer imagens do que letras ou números [3]. Geralmente, padrões invisíveis quando exibidos em tabelas ou em conjunto de dados são imediatamente percebidos quando apresentados de maneira sintetizada, na forma de gráficos e imagens [1]. Desta maneira, interfaces gráficas contendo mapas e gráficos que ilustram em tempo-real a evolução das variáveis de estado de um modelo ambiental podem tornar o processo de depuração desse modelo mais fácil e eficaz.

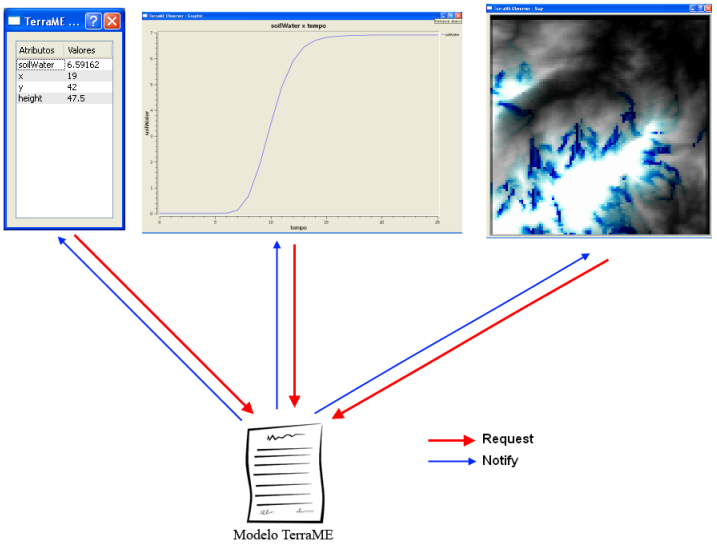

Figura 1. Padrão *Observer*, protocolo de comunicação e os observadores do tipo Tabela, Gráfico e Mapa (Adaptado de [4])

No ambiente de modelagem TerraME, as interfaces gráficas para monitoramento e análise dos modelos (*observer*) são implementadas na forma do Padrão *Observer* [4] onde cada interface gráfica, doravante chamada *observer*, é responsável por apresentar os dados de um *subject*. Um *subject* é qualquer entidade presente no modelo, cujos valores de suas propriedades determinam o estado corrente da simulação.

O TerraME[1] é um sistema para o desenvolvimento de modelos ambientais espacialmente-explícitos em múltiplas

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

escalas que permite o uso combinado de múltiplos paradigmas de modelagem, entre eles, teoria de agentes, teoria geral de sistemas ou teoria de autômatos celulares [5]. Ele foi desenvolvido em parceria pela Universidade Federal de Ouro Preto (UFOP) e pelo Instituto Nacional de Pesquisas Espaciais (INPE). Nesse sistema, vários *observers* podem ser acoplados a um único *subject*, para que o estado de uma entidade seja analisado de diferentes maneiras. Um *subject* ao ter seu estado alterado, notifica automaticamente todos os *observers* a ele aclopado. Um *observer* após receber essa notificação, requisita o novo estado do *subject* e passa a apresentá-lo. A Figura 1 apresenta a interação entre um *subject* e seus *observers*. Cada *observer* apresenta de maneira diferente os mesmos dados provenientes de um *subject*, neste caso, um componente de um modelo TerraME.

A comunicação entre *observers* e *subjects* deve acontecer de maneira eficiente, consumindo o mínimo de tempo de processamento, capacidade de armazenamento e capacidade de transmissão do canal de comunicação. Este trabalho objetiva desenvolver um protocolo e uma estrutura de dados que irão conferir eficiência à comunicação entre *subjects* e *observers* TerraME. Desta maneira, busca-se tornar viável o monitoramento e a análise, em tempo-real, da evolução da dinâmica de modelos ambientais espacialmente-explícitos em múltiplas escalas.

Na versão atual do Padrão *Observer* implementado na plataforma TerraME, o estado completo de um *subject* é serializado e transmitido a cada *observer* que o esteja monitorando, sem que nenhum processamento seja reaproveitado mesmo quando o mesmo estado precisa ser comunicado a diversos *observers*. Isto é, quando há dois ou mais *observers* associados a um mesmo *subject*, cada *observer* recupera o estado do *subject*, o utiliza e em seguida o descarta. Este processamento é realizado tantas vezes quanto for o número de *observers* lançados pelo usuário TerraME.

Neste trabalho, integrou-se o padrão de projeto *BlackBoard* [6] às estruturas de dados existentes no sistema TerraME. O *BlackBoard* age como uma área de memória *cache*, na qual os estados recuperados a partir dos *subjects* são armazenados temporariamente e podem ser reutilizados por outros observadores. Desta forma, busca-se reduzir o tempo de processamento envolvido na serialização dos estados, assim como a taxa de utilização do canal de comunicação entre *subjects* e *observers*. Os resultados obtidos mostram que o tempo de processamento para monitoramento dos modelos TerraME foi reduzido e que, no entanto, uma maior quantidade de memória é consumida para manutenção da *cache*.

O artigo está organizado da seguinte forma. Os sistemas computacionais utilizados são descritos nas Seção II. A Seção III apresenta a metodologia utilizada no desenvolvimento deste trabalho. As análises de complexidade são apresentadas na Seção IV. Os experimentos executados são descritos na Seção V. Ao final desse artigo, são apresentados a conclusão, Seção VII, e os trabalhos futuros, Seção VIII.

II. Sistemas Computacionais

TerraME

TerraME é um *framework* de modelagem e foi criado para avaliar o modelo de Autômatos Celulares Aninhados - Nested-CA [5], utiliza múltiplas escalas onde, para cada escala há uma resolução espacial, analítica e temporal. Esse ambiente permite descrever o modelo na linguagem C++ e na linguagem de modelagem TerraME. A linguagem TerraME estende a linguagem LUA [7]. LUA é uma linguagem interpretada, criada para ser estendida e prioriza cinco itens: portabilidade, simplicidade, pequeno tamanho, "acoplabilidade" e eficiência [8].

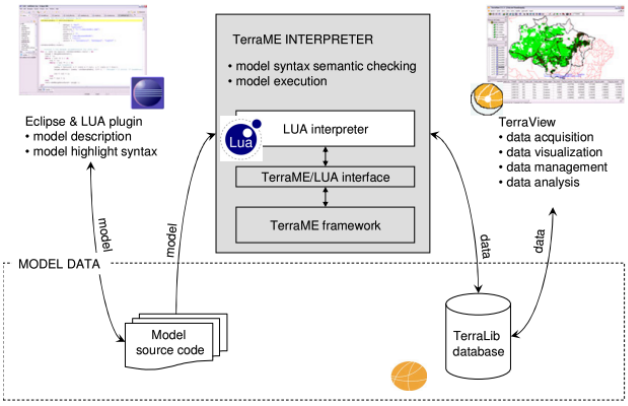

Figura 2. Módulo TerraME e serviços (Fonte: [5])

O ambiente de modelagem recebe como entrada um modelo escrito na linguagem TerraME, o interpretador do *framework* lê, interpreta o modelo e invoca as funções do ambiente TerraME (Figura 2). A biblioteca TerraLib[2] é usada para a leitura e escrita de dados resultantes da simulação e para prover mecanismos para o georeferenciamento de banco de dados espaciais. Após o término da simulação, os resultados podem ser visualizados, gerenciados e analisados usando do *software* TerraView[3] .

A arquitetura do ambiente de modelagem é apresentada na Figura 3. Na primeira camada, a TerraLib oferece gestão de dados espacial, serviços de análise e funções extras para a manipulação de dados temporais. Na segunda, o *framework* TerraME provê serviços para representação, simulação, calibração e validação de modelos ambientais. A terceira camada da arquitetura implementa a linguagem de modelagem TerraME e seu ambiente de execução. A interface TerraME / LUA estende a linguagem LUA com os novos tipos de dados espaciais de modelagem dinâmica e serviços para a simulação e avaliação do modelo. A última

[2]http://www.terralib.org
[3]http://www.dpi.inpe.br/terraview/

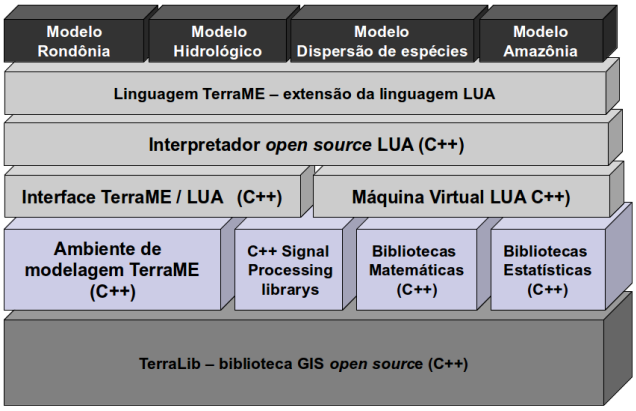

Figura 3. Arquitetura TerraME (Fonte: [5])

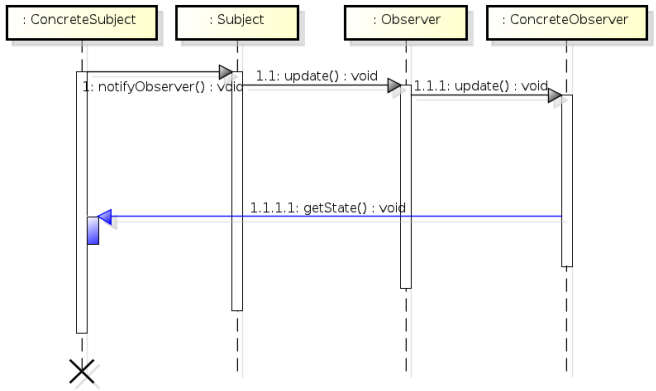

Figura 4. Diagrama de Sequência - Padrão *Observer*

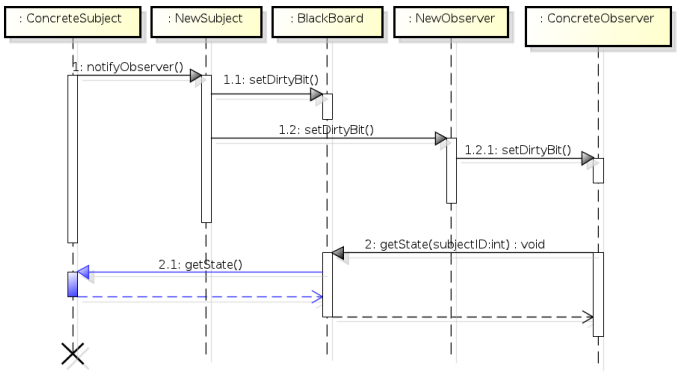

Figura 5. Diagrama de Sequência - Interação entre Padrão *Observer* e Padrão *BlackBoard*

camada é chamada camada de aplicação e inclui os modelos do usuário final.

Para um melhor entendimento dos experimentos, dois tipos de objetos em TerraME necessitam ser caracterizados:

- *Célula:* significa um local no espaço com propriedades, atributos e relações. Esse tipo pode ser visualizado pelos observadores do tipo Gráfico e Tabela;
- *Espaço celular:* é um conjunto de células e pode estar associado a um banco de dados geográfico. Esse tipo é observado apenas pelo observador do tipo Mapa.

A interface gráfica desses observadores são ilustrados na Figura 1.

III. Métodos

A. Padrões de Projeto

De acordo com Gamma *et al.* [4], padrões de projeto são soluções genéricas que podem ser adaptadas e aplicadas a diversas situações. A ideia principal é a reutilização, ou seja, problemas diferentes podem ser resolvidos através da adaptação e aplicação de uma mesma solução.

- *Observer:* Para visualizar de forma automática a evolução do modelo foi utilizado o Padrão *Observer*. Gamma *et al.* [4] o descreve como sendo um padrão comportamental e é usado para manter a consistência entre objetos. Quando algum atributo de um objeto *Subject* muda, ele notifica todos os objetos *Observer* associados a ele. Cada *observer*, após requisitar ao *subject* seu estado corrente, se atualiza e apresenta esse novo estado. A Figura 4 apresenta o diagrama de sequência desse padrão.
- *BlackBoard:* Segundo Buschmann *et al.* [6], é um padrão arquitetural. Pode ser visto como um repositório de dados, onde programas independentes trabalham cooperativamente sobre uma mesma estrutura de dados e montam sobre essa estrutura suas bases de conhecimento. Cada programa é especialista em resolver apenas uma parte do problema e juntos buscam encontrar uma solução. Essa solução pode ser parcial ou aproximada. O nome "BlackBoard" foi dado baseando-se em um exemplo real: especialistas trabalham juntos em frente a um quadro negro, cooperando e utilizando suas habilidades para resolver um problema. A cada instante, um deles se desloca até o quadro e utiliza suas habilidades em busca de uma solução para esse problema. O *BlackBoard* se difere de um repositório de dados por possuir uma estrutura interna de controle [6]. Esse conceito é utilizado neste trabalho pois, integrou-se o padrão *BlackBoard* ao padrão *Observer* e às estruturas de dados presentes em TerraME. O método de notificação do *subject* foi substituído por um método que define os dados do *BlackBoard* e dos *observers* como inválidos (Figura 5). O *observer*, ao ter seus dados definidos como inválidos, requisita ao *BlackBoard* os novos dados do *subject* observado. Caso o *BlackBoard* ainda não tenha informações válidas desse *subject*, requisita esses dados, armazena, redefine-os como dados válidos e repassa para o *observers*. Quando um outro *observer* requisita os dados, eles já estão presentes no *BlackBoard*. A nova versão da estrutura de dados é descrita no diagrama de classes (Figura 6). As classes em amarelo, representam o Padrão *Observer* e a classe em azul, o Padrão *BlackBoard*.

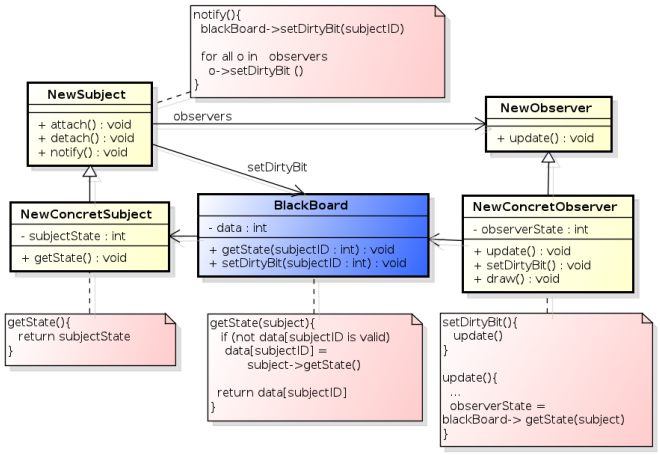

Figura 6. Diagrama de Classe - Integração entre Padrão *Observer* e Padrão *BlackBoard*

Algoritmo *Subject::getState(data)*

1 **para cada** $cell$ **em** $cellSpace$ **faça**
2 **para cada** $atributes$ **em** $cell$ **faça**
3 $data+ = [attributeType, attributeName, attributeValue]$
4 ...
5 **retorna** $data$

Algoritmo 1. Método *getState()* de um tipo *Subject*

IV. Análise de Complexidade

Nesta seção, descreveremos as análises de complexidade de tempo e de memória.

A. Complexidade de Tempo

Para a análise de complexidade vamos considerar apenas o custo do *observer* recuperar o estado do *subject*. A primeira versão nunca reutiliza os dados e cada observador é responsável pela gerência desses dados.

Desta forma, para a primeira versão, a complexidade de tempo é *O(n × m)*, onde *n* é a quantidade de células no espaço celular e *m* é a quantidade de atributos em cada célula.

O Algoritmo 1, apresenta o método *getState()* que é usado para recuperar os dados de um *subject*.

Para as novas versões, buscamos reduzir o número de chamadas ao método *getState()* do *subject*, devido ao grande custo que acarreta essa chamada. O Algoritmo 2 apresenta o método *getState()* presente no objeto *BlackBoard*.

A complexidade de tempo, usando o *BlackBoard*, no melhor caso é $O(1)$ e ocorre quando os dados estão presentes no *BlackBoard* e estão definidos como dados válidos. Entretanto, a complexidade para o pior caso, é *O(n × m)*. Este caso ocorre sempre que os dados devem ser recuperados do *subject*.

Algoritmo *BlackBoard::getState(subject)*

1 ...
2 **se** ($data[subjectId]$ não é válido) **então**
3 $data[subjectID] = subject\text{->}getState(aux)$
4 **retorna** $data[subjectID]$

Algoritmo 2. Método *getState()* do *BlackBoard*

B. Complexidade de Espaço

A complexidade de espaço na primeira versão não foi calculada pois, a estrutura de dados necessária é temporária. Cada *observer* requisita os dados ao *subject*, utiliza e descarta. Para a nova versão, a estrutura de dados é mantida durante toda a simulação. Portanto, a complexidade de espaço é *O(k × n × m)*, onde *k* é o número de observadores que não observam a mesma célula.

A complexidade de espaço na estrutura de dados, usado para recuperar um novo estado de um *subject*, no pior caso é *O(k × n × m)*. Entretanto, para a nova versão, o melhor caso ocorre quando um segundo observador de um mesmo *subject* requisita um dado já recuperado e armazenado no *BlackBoard*. Nesse caso, não há necessidade de recuperar os dados do *subject*.

Tabela I
DESCRITA DAS IMPLEMENTAÇÕES

Implementações	Padrão Utilizado	Estrutura de dados	Gerenciado
Versão 1	Observer	nenhuma	não
Versão 2	Observer + BlackBoard	Hash e Vector	não
Versão 3	Observer + BlackBoard	Hash e Vector	sim

Tabela II
EXPERIMENTOS 1

Experimentos	Número de Observadores
Teste 1	1
Teste 2	2
Teste 3	3

Tabela III
EXPERIMENTOS 2

Experimentos	Número de Observadores
Teste 1	100
Teste 2	200
Teste 3	400

V. Experimentos

Os experimentos foram realizados no seguinte ambiente:

- Processador: Pentium D 2,8GHz

- Memória: 3,2GB
- Sistema Operacional: Windows XP SP3

Os experimentos avaliaram o desempenho da simulação, o consumo de memória ao utilizar o *BlackBoard* e o comunicação gerada ao requisitar os dados de um *subject*.

Para avaliar o desempenho quanto ao uso de uma estrutura de dados, comparou a primeira versão e a nova versão que utiliza o padrão *BlackBoard*. Considerou-se também, para essa nova versão duas implementações, cada uma usando duas estruturas de dados distintas: o *Hash* e o *Vector*. Avaliou o *BlackBoard* como um repositório de dados e como definido por Buschmann [6], utilizando uma estrutura interna de controle. Essa estrutura de controle doravante chamada de gerencia, destrói os dados armazenado e é executada em intervalos de 20 segundos. Quando um *observer* requisitar esses dados e eles não estiverem presentes na estrutura, eles são inseridos novamente. Implementou-se duas versões, uma com gerencia interna e outra sem. Desta forma, as versões ficaram como descrito na Tabela I. Cada teste foi repetido cinco vezes e ao término de cada um, foi recolhido o tempo de execução da simulação e o espaço utilizado pela aplicação. Em uma simulação real, um intervalo de vinte segundos para um modelo pode representar anos ou até séculos, para outro, pode representar apenas alguns segundos.

Executou-se dois experimentos cada um com três testes incrementais (Tabela II e Tabela III). Nesses experimentos utilizou um modelo de drenagem que considera um espaço celular composto de dez mil células. A simulação desse modelo avalia sessenta iterações.

No primeiro experimento, avaliou o tipo composto Espaço Celular, no qual algum atributo das células pertencentes a esse espaço celular é avaliado. Esse tipo é visualizado pelo observador do tipo Mapa. No primeiro teste, foi observado em um mesmo *observers* os atributos quantidade de água e altimetria. No segundo, adicionamos um outro *observer* para monitorar, isoladamente, o atributo quantidade de água. No terceiro, além dos *observers* dos testes anteriores, um outro foi adicionado para monitorar apenas o atributo altimetria.

No segundo experimento, avaliou o tipo Célula, esse pode ser visualizado pelos observadores Gráfico e Tabela. Nesse experimento, foi observado os atributos quantidade de água e altimetria. No primeiro teste desse experimento, foi utilizado cem *observers* (50 do tipo tabela e 50 do tipo gráfico) associados a cinquenta células distintas. No segundo teste, usou-se duzentos *observers* (100 do tipo tabela e 100 do tipo gráfico) observando cem células diferentes. No terceiro, usamos quatrocentos observadores (200 do tipo tabela e 200 do tipo gráfico) monitorando duzentas células distintas.

VI. Resultados

Os resultados dos experimentos são descritos a seguir.

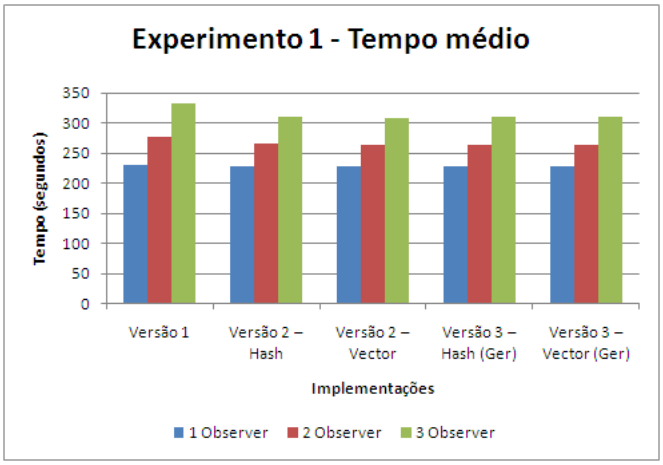

Figura 7. Experimento 1 - Tempo médio

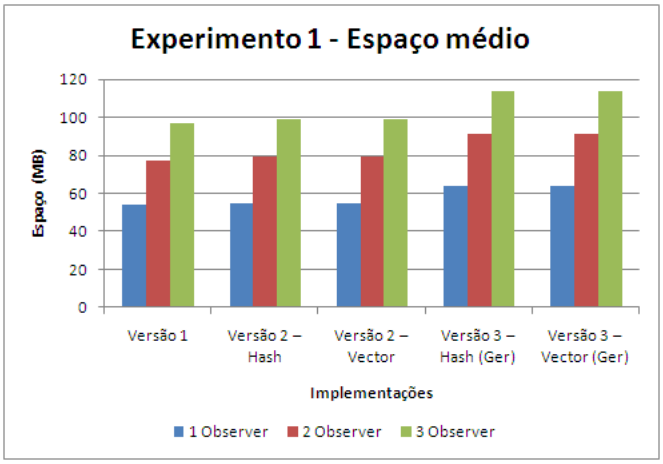

Figura 8. Experimento 1 - Espaço médio

Tabela IV
Experimento 1 - Teste 1

Implementações	Valores Médios		Coef. Variação (%)	
	Tempo (s)	Espaço(MB)	Tempo	Espaço
Versão 1	229,0	53,841	0,39	0,17
Versão 2 - Hash	227,6	54,960	0,53	0,39
Versão 2 - Vector	226,6	54,848	0,45	0,18
Versão 3 - Hash (Ger)	227,2	63,798	0,76	0,08
Versão 3 - Vector (Ger)	227,0	63,820	0,39	0,09

Tabela V
Experimento 1 - Teste 2

Implementações	Valores Médios		Coef. Variação (%)	
	Tempo (s)	Espaço(MB)	Tempo	Espaço
Versão 1	275,8	77,137	0,15	0,09
Versão 2 - Hash	264,6	79,101	0,39	0,08
Versão 2 - Vector	263,2	79,011	0,50	0,15
Versão 3 - Hash (Ger)	264,0	90,891	0,24	0,10
Versão 3 - Vector (Ger)	263,0	91,538	0,00	0,10

Tabela VI
EXPERIMENTO 1 - TESTE 3

Implementações	Valores Médios		Coef. Variação (%)	
	Tempo (s)	Espaço(MB)	Tempo	Espaço
Versão 1	333,0	96,681	0,27	0,11
Versão 2 - Hash	310,6	99,050	0,44	0,12
Versão 2 - Vector	308,4	99,077	0,16	0,10
Versão 3 - Hash (Ger)	310,0	113,921	0,43	0,09
Versão 3 - Vector (Ger)	308,8	113,862	0,32	0,10

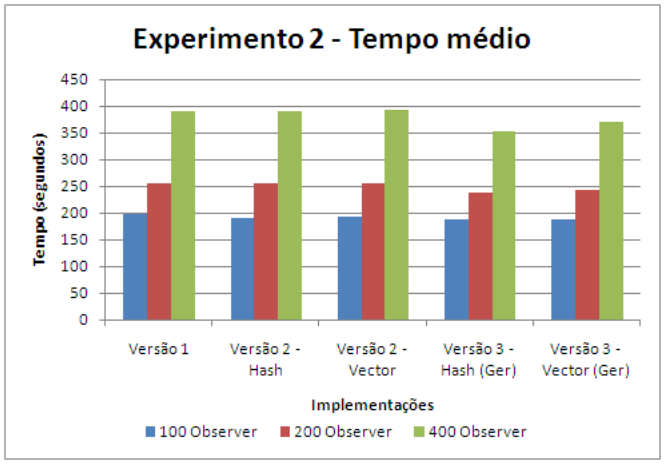

Figura 9. Experimento 2 - Tempo médio

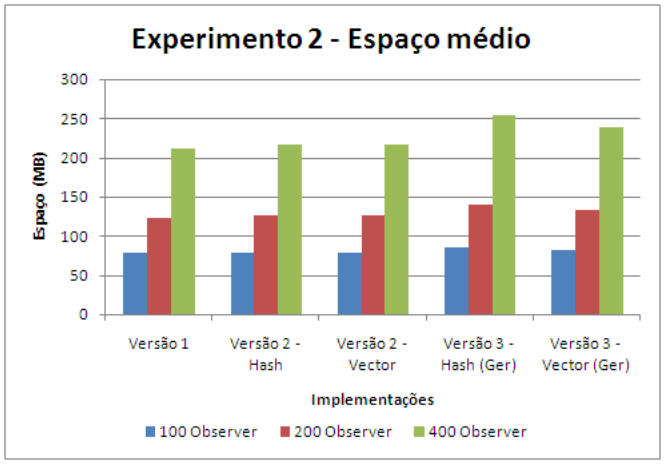

Figura 10. Experimento 2 - Espaço médio

Tabela VII
EXPERIMENTO 2 - TESTE 1

Implementações	Valores Médios		Coef. Variação (%)	
	Tempo (s)	Espaço(MB)	Tempo	Espaço
Versão 1	197,8	77,634	0,38	0,11
Versão 2 - Hash	190,0	78,634	0,74	0,27
Versão 2 - Vector	191,6	78,825	0,53	0,21
Versão 3 - Hash (Ger)	186,4	84,436	0,55	0,16
Versão 3 - Vector (Ger)	188,6	81,708	0,54	0,17

Tabela VIII
EXPERIMENTO 2 - TESTE 2

Implementações	Valores Médios		Coef. Variação (%)	
	Tempo (s)	Espaço(MB)	Tempo	Espaço
Versão 1	255,2	123,354	0,76	0,18
Versão 2 - Hash	256,2	125,594	0,29	0,21
Versão 2 - Vector	256,2	125,549	0,67	0,17
Versão 3 - Hash (Ger)	237,8	140,474	0,67	0,19
Versão 3 - Vector (Ger)	242,2	132,736	0,31	0,18

Tabela IX
EXPERIMENTO 2 - TESTE 3

Implementações	Valores Médios		Coef. Variação (%)	
	Tempo (s)	Espaço(MB)	Tempo	Espaço
Versão 1	391,0	211,270	0,84	0,13
Versão 2 - Hash	391,2	215,764	0,78	0,07
Versão 2 - Vector	392,4	215,866	0,44	0,24
Versão 3 - Hash (Ger)	351,7	254,517	0,39	0,37
Versão 3 - Vector (Ger)	370,4	238,043	0,50	0,24

Experimento 1: Os resultados do primeiro teste desse experimento são apresentados na Tabela IV. Para o monitoramento com apenas *observer*, o uso de uma estrutura de dados reduziu o tempo de simulação em alguns segundos, no entanto, foi necessário utilizar mais memória.

Os resultados do segundo teste são apresentados na Tabela V. Um melhor desempenho da simulação é apresentado nesse teste. Obteve-se uma redução de, aproximadamente, 10 segundos no tempo de simulação. Porém, o consumo de memória teve um acréscimo em torno de 15 MB.

Os resultados do terceiro teste são apresentados na Tabela VI. Para esse teste, pode-se perceber que o uso da estrutura contribuiu para um ganho de desempenho. Mas em contra partida, foi necessário uma maior quantidade de memória consumida para manter esses dados.

As Figuras 7 e 8 apresentam graficamente os resultados desses experimento.

Experimento 2: Os resultados do primeiro teste desse experimento são apresentados na Tabela VII. Percebe-se que houve um ganho no desempenho da simulação e maior consumo de memória nas versões 2 e 3.

Os resultados do segundo teste são apresentados na Tabela IX.. Nesse teste as versões 2, usando *Hash* e *Vector*, tiveram um desempenho pior comparado com a versão 1. Essa perda de desempenho necessita ser investigada. No entanto, o desempenho e o consumo de memória na versão 3, usando *Hash* e *Vector* é bem superior os demais.

Os resultados do terceiro teste são apresentados na Tabela VI. Como ocorreu no teste 2, o desempenho utilizando a versão 3 da estrutura se demonstrou eficaz que as demais e exigiu um maior consumo de memória. A perda de desempenho apresentada na versão 2 será investigada nos trabalhos futuros.

As Figuras 9 e 10 apresentam graficamente os resultados desses experimento.

VII. Conclusão

O uso de um estrutura de dados melhorou o tempo de monitoramento da simulação. No entanto, o consumo de memória foi incrementado. Esse melhor desempenho foi obtido porque não houve necessidade de requisitar alocação de memória a cada atualização do *observer* e também, para os casos em que há mais *observers* associados a um mesmo *subject*, apenas a primeira requisição solicita diretamente ao *subject* os dados, para as demais requisições, o dado já esta presente no *BlackBoard*.

VIII. Trabalhos Futuros

As seguintes questões serão consideradas no trabalhos futuros:

- Aprimorar a gestão do *BlackBoard*, baseando-se em políticas de cache.
- Considerar a recuperação apenas dos dados modificados.
- Avaliar a possibilidade de utilizar programação paralela para recuperação dos dados do modelo.
- Investigar a perda de desempenho ao utilizar um grande número de *observers*

Referências

[1] T. A. DeFanti, M. D. Brown, and B. H. McCormick, "Visualization: Expanding scientific and engineering research opportunities," *IEEE Computer Society Press*, vol. 22, pp. 12–25, August 1989.

[2] C. Kelleher and T. Wagener, "Ten guidelines for effective data visualization in scientific publications," *Environmental Modelling & Software*, vol. 26, no. 6, pp. 822 – 827, 2011.

[3] K. Cukier, "A special report on managing information: Data, data everywhere," *The Economist*, feb 2010. [Online]. Available: http://www.economist.com/node/15557443?story_id=15557443

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[5] T. G. S. Carneiro, "Nested-ca: um fundamento para a modelagem de uso e cobertura do solo em múltiplas escalas," Tese de doutorado, INPE - Instituto Nacional de Pesquisas Espaciais, Brazil, Computação Aplicada, Jun. 2006.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., 1996.

[7] R. Ierusalimschy, "A linguagem lua," 2007, http://www.inf.puc-rio.br/ roberto/talks/ufrj-2007.pdf, visitado em 05/07/2011.

[8] ——, *Programming in Lua, Second Edition*. Lua.Org, 2006.