TerraME Observer e GIMS:
Novas Demandas e Futuras Possibilidades

Pedro Ribeiro de Andrade
06/05/2011

Este relatório faz uma análise detalhada do TerraME-Oberver e do TerraME-GIMS, apontando deficiências da versão 1.0 BETA e indicando novas demandas e futuras possibilidades de desenvolvimento. Várias das considerações aqui descritas não apresentam soluções, sendo apenas questionamentos que devem resultar em novas reflexões para as futuras versões.

Legend

A última versão da documentação das Legendas, usada neste texto, é a seguinte:

Function	
Legend	Class that defines a legend to be used in an observer. It is used only when the observer is of type OBSERVERS.MAP. The configuration of a legend can be changed visually within the graphical interface along the simulation. **type:** the type of the attribute to be observed. It has to be one of TYPES.BOOL, TYPES.NUMBER, TYPES.DATETIME, TYPES.TEXT. For each of these types, some other parameters are compulsory. **groupingMode:** how the data is going to be sliced. It has to be one GROUPING.EQUALSTEPS, GROUPING.QUANTIL, GROUPING.STDDEVIATION, GROUPING UNIQUEVALUE. **slices:** the number of colors to be used for plotting. **precision:** the number of decimal digits for slicing. **stdDeviation:** when the grouping mode is stddeviation, it has to be one of STDDEVIATION.FULL, STDDEVIATION.HALF, STDDEVIATION.QUARTER. STDDEVIATION.NONE is the default value for it. **maximum:** the maximum value of the attribute (used only for numbers). **minimum:** the minimum value of the attribute (used only for numbers). **colorBar1:** a table where each position is also a table with the color. In the case of unique value, each position needs to have also a value of the respective attribute. **colorBar2:** a table similar to the previous parameter. It is needed only when the standard deviation is the chosen strategy.

A construção de Legendas precisa ser simplificada, principalmente através de uma análise mais profunda dos valores default. Peguemos como exemplo a legenda a seguir:

```
agentsLeg = Legend{
       type = TYPES.TEXT,
       groupingMode = GROUPING.UNIQUEVALUE,
       slices = 3,
       precision = 5,
       stdDeviation = STDDEVIATION.NONE,
       maximum = 1,
```

```
        minimum = 0,
        colorBar1 = {
                {YELLOW, "Sheep"},
                {RED, "Wolf"}
        }
}
```

Várias considerações podem então ser feitas a respeito dos valores default:

1. o type de uma legenda está intimamente ligado ao atributo que será usado para plotagem, sendo que a partir dos valores de um atributo podemos extrair o seu tipo, fazendo este atributo type desnecessário na descrição de uma legenda. Na pior das hipóteses, o type poderia ser deduzido do tipo dos valores de colorBar.
2. mais um motivo para usar strings ao invés de CONSTANTES: em type, o usuário poderia usar os mesmos valores restantes da função type( ) do Lua, que são "boolean", "number" e "string" (únicos tipos que interessam a esta função).
3. groupingMode também depende do type, sendo que para tipos string o único groupingMode plausível é UNIQUEVALUE.
4. Slices, precision, stdDeviation, maximum e minimum são desnecessários quando se está usando UNIQUEVALUE. Neste caso, o número de slices é exatamente o número de elementos da colorBar.
5. O colorBar poderia ter três (ao invés de dois) elementos na lista, sendo que o terceiro seria o texto que aparecerá na interface gráfica, tendo como valor default o segundo valor. Exemplo:

```
colorBar1 = {
        {WHITE, 0, "Water"},
        {RED, 1, "Burned"},
        {GREEN, 2} -- nao usando o terceiro, o valor mostrado sera 2
    }
```

6. O uso de colorBar1 e colorBar2 vem da interface gráfica do TerraView. Como o modelos descritos no TerraME usam código-fonte, a descrição dos elementos não deve necessariamente seguir a mesma estratégia adotada pelo TerraView. Deve ser repensado se o uso de dois objetos colorBar é realmente necessário, ou se apenas uma colorBar não seria suficiente para descrever desvios padrão.

TerraME-Observer

A última versão da documentação do Observer, usada neste texto, é a seguinte:

Function	
Observer	Observer is a strategy used for collecting data from the objects of a model in order to save or to graphically plot them. In fact, it is not a class as the other ones, but some of TerraME objects have a function createObserver whose parameters are described here. Currently there are six types of legends. The function createObserver can receive a couple of parameters according to the strategy to be used for observing the entity of the model. The other lines below describe the values to be used according to the first parameter, which is the type of the legend.
OBSERVERS. DYNAMIC	Observer that creates a dispersion chart showing the variation of an attribute of a Cell, Agent, Automata, or Environment in time.

GRAPHIC	**Second argument:** a table with one string containing the attribute name to be used for plotting. The value represents the y axis and the time is used as x. Therefore, notifyObservers(modelTime) has to be called with the time as argument. **Third argument:** a table with four strings. The four values are the following: title of the graphic, name of the curve, name of x axis, and name of the y axis.
OBSERVERS. GRAPHIC	Observer that creates a dispersion chart with two attributes of a Cell, Agent, Automata, or Environment. **Second argument:** a table with two strings containing the attribute names to be used for plotting. The first value represents the y axis and the second, x. When only one attribute is passed, x is represented by the time. **Third argument:** a table with four strings. The four values are the following: title of the graphic, name of the curve, name of x axis, and name of the y axis.
OBSERVERS. LOGFILE	Observer that saves the attributes of a Timer, Event, Agent, Automaton or Environment to a text file in a tabular format. **Second argument:** a table with strings containing the attribute names to be saved. When empty, the observer will use every available attribute of the object(s). **Third argument:** a table with three strings, in the following order: name of the file to be written, the attribute separator character (i.e., ";"), and the open mode, "w" for writing a new file, "w+" for overwriting an existing file, or "a" to append an existing file.
OBSERVERS. MAP	Observer that creates a map with the spatial distribution of a given attribute of a CellularSpace, Agent, Automaton, or Trajectory. **Second argument:** a table with one or two strings containing the attribute names to be used for plotting. The first value represents the front layer and the second one (optional) is the background layer. **Third argument:** a table with one Legend for each layer to be plotted. See Legend type for details.
OBSERVERS. TIMER	Observer that creates a display with the current time and event queue of a given timer.
OBSERVERS. TABLE	Observer that creates a display with the current attributes of a Timer, Event, Agent, Automaton or Environment. **Second argument:** a table with strings containing the attribute names to be saved. When empty, the observer will use every available attribute of the object(s). **Third argument:** a table with one string for each attribute used in the second argument, representing their labels.
OBSERVERS. TEXTSCREEN	Observer that creates a display in a tabular format with the attributes of a Timer, Event, Agent, Automaton or Environment. The arguments are Same as OBSERVERS.TABLE.
OBSERVERS. UDPSENDER	Observer that sends the attributes of any TerraME object except Message, State, Jump and flow, to a UDP port of a given IP host.

Exemplos de Observers descritos na mesma documentação:

```
cell:createObserver(
    OBSERVERS.DYNAMICGRAPHIC,
    {"soilWater"},
    {"title",
      "soilWater (mm)",
      "time (s)",
      "soil water"}
)
```

```
cell:createObserver(
    OBSERVERS.GRAPHIC,
    {"soilWater", "temperature"},
    {"title",
      "water (mm)",
      "temp (K)",
      "soil water"}
)
```

```
cs:createObserver(
    OBSERVERS.MAP,
    {"soilWater", "height"},
    {soilWaterLeg, heightLeg}
)
```

A sintaxe adotada para a descrição de um Observer na atual versão é consideravelmente longa, o que facilita a ocorrência de erros advindos do processo de codificação. Existe um excesso de uso de chaves durante a descrição dos parâmetros das funções createObserver (por exemplo '{}, {"agent.csv"}' para o LOGFILE), e os nomes das estratégias, tais como "OBSERVERS.DYNAMICGRAPHIC" e "OBSERVERS.TEXTSCREEN", são grandes (pelo menos 20 caracteres cada um) e vários argumentos não possuem valores default (ou pelo menos não está bem documentado). Se estas constantes fossem tratadas como string, e não como números inteiros, o próprio TerraME poderia ter o controle sobre eventuais erros na escrita destes elementos. Adicionalmente, se os parâmetros usados para a criação dos Observers fossem nomeados (e não ordenados) da mesma forma que as classes consolidadas do TerraME, tal como CellularSpace, o código seria mais simples e legível. Experiências relativas a funções de plotagem em outros softwares tais como R (www.r-project.org) mostram que está é uma estratégia viável. Quase todos os parâmetros poderiam ter valores default, que seriam dependentes da classe do objeto para qual o Observer será criado. Por exemplo, um Observer para um Timer teria como estratégia default o OBSERVER.SCHEDULER, para um CellularSpace seria OBSERVER.MAP, e assim sucessivamente).

Sugestões gerais para futuras implementações do Observer:

1. É necessário trabalhar com mais atenção algo que se iniciou para resolver um problema específico, o "currentState", que descreve o estado atual de um Agent ou Automaton, e também aproveitar a estratégia usada para permitir novas possibilidades. A observação deste estado é possível a partir do OBJECT.TEXTSCREEN, mas não funciona com outros Observers:

   ```
   ag1:createObserver(OBSERVERS.GRAPHIC,{"cont", "currentState"},{""})
   ag1:createObserver(OBSERVERS.DYNAMICGRAPHIC,{"currentState"},{"", "", "", ""})
   ```

   O atributo "currentState" não é visível por parte do usuário Lua (e talvez nem na camada C++, pois nem todos os Observers conseguem tratá-lo como atributo). O único acesso a este valor que o usuário Lua possui é através da função getStateName(). Uma solução para deixar mais transparente o acesso a valores

deste tipo seria possibilitar o modelador passar funções dos objetos como strings e o próprio Observer ficaria encarregado de verificar se as strings passadas como atributos são valores ou funções. Caso sejam funções, ele executa a função e usa o seu retorno como se fosse o valor do atributo. Isto é eficiente por ser resolvido em tempo de construção do objeto. Adicionalmente, na implementação de tabelas em Lua, uma função também é considerado um atributo da tabela. Por exemplo, o código a seguir poderia funcionar de acordo com esta estratégia:

```
ag1:createObserver(OBSERVERS.GRAPHIC, {"cont", "getStateName"}, {""})
```

2. O código a seguir mostra um exemplo de criação de um Observer que usa um mapa resultante de um outro Observer para se observar o comportamento de um Agent no espaço.

```
obs1 = cs:createObserver(OBSERVERS.MAP, {"cover"}, {coverLeg})
ag:createObserver(OBSERVERS.MAP, {"currentState"}, {cs, obs1, agentsleg})
```

   Durante a criação do Observer para o agente, é necessário informar o tipo de observer (MAP), mesmo sabendo que será usado o resultado da observação de um outro Observer (obs1) como plano de fundo para o resultado da observação. Desta forma, pode-se considerar que o uso de MAP não adiciona qualquer semântica à linha de código, uma vez que a estratégia poderia ser resolvida apenas pela presença do objeto obs1 como parâmetro.
3. A versão atual do Observer não possui um ferramental para controlar a execução da simulação. Tanto que, ao final da execução do modelo, o usuário tem que fechar a tela do prompt do DOS para finalizar a simulação. Outra estratégia poderia ser fazer com que o TerraME, quando executado de fora do DOS (por exemplo no Crimson), não mostre nada na tela, de forma que tudo o que é visto pelo usuário seja feito através de janelas gráficas com o Observer. Isso é possível?
4. A qualidade das interfaces gráficas produzidas pelo Observer é relativamente baixa, com vários espaços sobrando nas bordas e eixos mal delineados. É necessário despender algum esforço no melhoramento destas interfaces.
5. O Observer do Timer continua com problemas na visualização de Eventos com diferentes períodos (demo23_ObserverTimer.lua).
6. A arquitetura do Observer possui dois pontos de entrada para o modelador. O primeiro é a construção do Observer propriamente dito, com sua representação visual e os atributos a serem usados (caso necessário). O segundo se refere às mensagens enviadas para o Observer no decorrer da simulação para atualizar a janela gráfica com novas informações dos últimos passos da execução do modelo. Uma tentativa de se automatizar o envio de mensagens para o Observer, definindo-se eventos padrões que podem ser disparados automaticamente sem um maior controle por parte do modelador pode ser interessante. Exemplos podem ser citados: um Timer que atualiza a sua interface cada vez que um evento é executado ou a cada novo tempo de simulação, ou um agente que atualiza a interface gráfica que descreve o seu estado toda vez que o mesmo é alterado ou se redesenha quando alguma das suas relações com o espaço é alterada. Isto poderia se

transformar em mais um parâmetro na construção do Observer. Obviamente que ainda assim será necessário manter a estratégia atual de envio de mensagens caso o modelador queira ter um maior controle sobre a atualização dos Observers. Entretanto, este tipo de estratégia pode facilmente se materializar em representações a serem usadas pelo GIMS. Um relatório que estude mais profundamente esta estratégia e liste os possíveis eventos é essencial. Algumas idéias podem ser obtidas com os Watchers do RePast.

7. Implementar Observers para objetos do tipo Society, que seriam simples Observers para coleções de agentes simultaneamente. Estes Observers executariam os Observers dos Agentes que eles contêm dentro de um forEach. O mesmo seria aplicado ao notifyObservers. Um Observer de Society deveria também gerenciar a questão da criação e remoção de agentes durante a execução do modelo. Verificar também como fazer para ter apenas um Observer em memória para toda a Society, ao invés de ter um para cada agente individualmente.
8. Possibilitar o modelador descrever o visual de um agente de acordo com o estado do mesmo. Uma solução pode ser o State possuir um atributo "icon" que aponta para um arquivo de imagem. Quando o agente estiver em um determinado estado, ele é desenhado com o ícone associado a este estado em específico. Legendas podem estar associadas a (i) os símbolos de cada estado ou (ii) cores usadas para desenhar os estados.
9. A cor do desenho dos Agents não é a mesma cor definida na legenda. Aparentemente existia algum efeito relacionado à transparência, pois a cor de desenho tende a se aproximar da cor de fundo, mas quando o checkbox foi *desativado* a verdadeira cor da legenda foi usada para desenhar os agentes. Quando o mesmo checkbox foi reativado a cor voltou a ficar errada. Adicionalmente, o combobox abaixo de Operations não possui um significado claro e precisa ser documentado.
10. Quando dois Agents estão em uma mesma célula, o Observer atualmente desenha eles de forma sobreposta, causando a impressão de que existe apenas um agente dentro de uma determinada célula. Novas estratégias são necessárias para se desenhar agentes dentro de células, como por exemplo distribuir aleatoriamente o centro do agente, ou preencher a célula do canto superior esquerdo até o canto inferior direito em linhas horizontais/verticais. Isto vai requerer uma diminuição no tamanho do ícone usado para desenhar os agentes.
11. O Observer está sempre ligado a uma Legend descrita pelo modelador. Devem ser estudadas maneiras de enfraquecer esta ligação. Por exemplo, não seria uma boa idéia usar a legenda do próprio TerraView com o seu atributo selecionado e suas cores definidas como sendo a legenda default no TerraME?
12. Existe um problema no início da execução do Observer para espaços celulares, não desenhando o mapa na tela conforme seu tamanho real. Apenas quando o Observer é ativado pela primeira ou segunda vez o desenho se encaixa ao real tamanho da janela. Isto não deveria acontecer. Adicionalmente, existem problemas nas bordas dos espaços celulares, cujas primeiras e últimas linhas e colunas não são desenhadas corretamente.

13. Deveria ser possível desenhar as linhas separando as células de um espaço celular, tal como é feito no TerraView. Isto será particularmente útil quando puder desenhar mais de um agente por célula.
14. Mesmo com o modelo do Anexo I funcionando perfeitamente, uma mensagem de erro é retornada (mesmo não afetando o resto da simulação): 'Error: The "currentState" attribute has been configured incorrectly. An error might have occurred when this attribute was defined in the legend.', além de mostrar o conteúdo da pilha Lua, o que possui considerável potencial de assustar o usuário.
15. Outros Observers necessários estão relacionados a Neighborhoods e Trajectories. Várias estratégias de plotagem podem ser imaginadas para estas duas classes.
16. Como fazer para implementar o conceito de Observer no GIMS? O Observer hoje se materializa na camada Lua do TerraME como uma função, e não como um tipo, sendo necessário uma implementação desta função para cada tipo do TerraME. Desta forma, incluir ele como um componente a mais não expressa efetivamente o que ocorre no TerraME. Por que não ter o Observer como um tipo Lua, ao invés de uma função? Assim, teríamos apenas um construtor que receberia como parâmetro o objeto a ser observado. Apenas mais uma observação: o Observer, que é mais complexo e importante do que a Legenda (que é um parâmetro na construção de Observers), não é uma classe, enquanto que a própria Legenda é.

TerraME-GIMS

Todos os testes da versão 1.0 BETA funcionaram com sucesso, apesar da dificuldade de se desenvolver modelos, visualizar e executar o código gerado. A implementação desenvolvida apresenta novos conceitos para a descrição visual de modelos no TerraME, atendendo aos requisitos descritos no projeto. Entretanto, uma futura análise da amigabilidade das interfaces gráficas implementadas será necessária, visto que estas ferramentas tem como objetivo facilitar o processo de desenvolvimento de modelos, tanto na sua construção quanto na observação dos resultados durante a simulação, principalmente para usuários que não possuem uma vasta experiência em ferramentas computacionais.

Sugestões gerais para futuras implementações do TerraME-GIMS:

1. O processo de instalação de todos os softwares, complementos e configurações é relativamente complicado e demorado. Adicionalmente, novas versões de software podem surgir com diferentes interfaces de usuário, tornando o processo ainda mais obscuro. Por exemplo, na primeira tentativa de instalação do plugin Lua obteve-se a seguinte mensagem de erro: "Cannot complete the install because some dependencies are not satisfiable org.keplerproject.ldt.feature_x64.feature.group [1.2.0.200802220021] cannot be installed in this environment because its filter is not applicable." Esta era a instalação do Plugin Lua versão 1.2, que não funcionou para o Eclipse SDK Helios 3.6.2 no Windows Seven (a versão 1.1 funcionou sem problemas, mas para um usuário sem experiência isto pode determinar a desistência do processo de instalação do software). Como o possível usuário do TerraME-GIMS não possui um forte conhecimento e facilidade de se trabalhar com informática, certamente vale a pena estudar a viabilidade da distribuição de uma

versão do Eclipse com o TerraME-GIMS previamente instalado e configurado, ou de um programa que configure o Eclipse automaticamente, possivelmente instalando até o próprio TerraME durante a própria instalação do GIMS. Assim, a interface do Eclipse poderia ser melhor adaptada ao modelador TerraME, removendo-se várias abas e botões desnecessários (a aba "Javadoc", que pode apenas fazer com que o modelador suponha que ele deve programar em Java, o botão "Install modeling components", que possui o desenho similar ao ícone usado para representar o Environment, e vários outros), reduzindo consideravelmente a complexidade da interface gráfica do Eclipse.

2. Não são feitas verificações referentes a se os nomes das variáveis são possíveis (por exemplo, nenhum objeto pode ter o nome começando com número ou possuindo caracteres especiais, ou usar nomes de classes ou de algum outro objeto no mesmo escopo).
3. Não existe qualquer indicação visual de que os objetos estão inconsistentes. Por exemplo, uma função que ainda não foi implementada deveria possuir alguma indicação de que existem pendências relativas a este objeto, um CellularSpace que não possui os parâmetros de configuração determinados deveria ser indicado como inconsistente. Antes de executar algum modelo, o GIMS deveria possuir uma funcionalidade de verificação de pendências, como parâmetros obrigatórios e conexões necessárias entre objetos, de forma a avisar o modelador sobre possíveis erros que podem acontecer durante a execução do modelo, indicando visualmente os componentes que precisam ser alterados.
4. Não está claro, no código gerado ou em possíveis aplicações futuras, qual seria a necessidade de se ter uma variável "id" dentro de cada objeto, com o mesmo nome do objeto.
5. Na "Pallette," existe um botão de "Note - create Note". Este botão descreve notas (em amarelo, finalmente uma cor!) que podem pertencer a alguns componentes. Não ficou claro porque apenas alguns componentes podem receber notas (por exemplo, CellularSpace). Estas notas poderiam aparecer no código-fonte na forma de comentários, e a futura engenharia reversa geraria comentários como notas.
6. Por que a visualização do Environment, diferentemente de todos os outros conceitos, possui dois retângulos, sendo que é possível criar novos objetos apenas no retângulo inferior? Caso seja necessário por causa de alguma restrição do Eclipse, o retângulo superior poderia ao menos ter um tamanho bem reduzido.
7. Alguns objetos não podem ser criados dentro de outros objetos, por restrições hierárquicas do próprio TerraME. Entretanto, após a criação destes objetos, os mesmos podem ser arrastados para cima de outros objetos, o que pode criar uma falsa impressão de hierarquia (por exemplo um Timer acima de um CellularSpace). O ato de arrastar deveria ter a mesma semântica do ato de criar, não permitindo este tipo de situação, o que evita interpretações visuais errôneas.
8. O atributo "database" do CellularSpace deveria ter a opção de o usuário selecionar o banco de dados a partir de um visualizador de arquivos do computador, ao invés de forçar o usuário escrever todo o caminho do arquivo manualmente.

9. Na tela principal, existe a função main, explícita e fixa. Sobre esta tela, é possível criar um, e apenas um, Environment. Por que não se ter então um Environment logo na tela principal, e então possibilitar ter vários Environments dentro deste?
10. O ícone Functions→Main não pode ser criado em qualquer ponto da interface. Na verdade, ele pode ser criado apenas se o usuário se confundir e apagar acidentalmente o antigo main. Por que não evitar que o original seja apagado, e então remover este ícone da barra de componentes?
11. É necessário implementar um algoritmo para auto-organizar os componentes visualmente de forma a eles, bem como as suas conexões, não se sobreponham uns aos outros. O algoritmo de "arrange all" disponibilizado pelo Eclipse não leva em consideração as especificidades dos objetos, como por exemplo dispor visualmente primeiro o espaço, depois o comportamento, e finalmente o tempo (da esquerda para a direita, de cima para baixo). Da mesma forma, os eventos do timer poderiam ser ordenados pelo seus tempos iniciais.
12. Os eventos do Timer poderiam ficar ordenados automaticamente da esquerda para a direita de acordo com o início da execução de cada um deles, independentemente do modelador, mas deveria ser possível rearranjá-los de acordo com outros atributos, como por exemplo as suas prioridades.
13. Na paleta que descreve os atributos de um componente ("Properties") do Eclipse, é possível alterar as cores ou estilos das fontes dos nomes de atributos? Como alguns atributos são obrigatórios, eles poderiam ser mostrados com mais ênfase (talvez em negrito) do que os atributos opcionais.
14. Os diagramas elaborados para cada classe do TerraME possuem pouca semântica. O simples nome do objeto não é suficiente para simbolizar o que este faz. Um sumário dos objetos seria interessante. Funções poderiam mostrar pelo menos os seus parâmetros; Agents poderiam mostrar pelo menos o número de estados, para que o modelador possa ter alguma informação sobre o objeto sem precisar abrir o diagrama de estados; CellularSpaces poderiam mostrar pelo menos o número de células e os nomes dos seus atributos. Os componentes poderiam ter também cores associadas, de forma a facilitar a identificação visual dos mesmos.
15. Excluindo-se a disposição hierárquica, não existe qualquer relação visual entre os objetos de um modelo. Por exemplo, um Timer que ativa um Agent deveria ser visualizado como uma flecha saindo do Timer em direção ao Agente. O mesmo se aplica quando o main ativa determinado Environment, ou quando um Agent possui um Trajectory para um CellularSpace. Novas possibilidades podem ser imaginadas.
16. Não existem funcionalidades no GIMS relativas à construção e o carregamento de vizinhanças. Estas poderiam ser criadas a partir de um novo componente que representaria as flechas descritas no item anterior. Com este componente, ao se clicar duas vezes em um CellularSpace, o GIMS poderia possibilitar o usuário a criar vizinhanças ou carregá-las de um banco de dados (o próprio GIMS poderia verificar se existem vizinhanças armazenadas em disco). Este conceito pode ser estendido para outros tipos de relação. Clicando em dois CellularSpaces distintos, temos uma relação espacial entre escalas. Clicando em um Timer e depois em um Agent, temos a criação de um evento para ativar o Agent. Clicando em um Agent e depois em um CellularSpace, temos a criação de um Trajectory. Clicando em dois

Environments distintos temos a integração entre escalas a partir das definições dos seus feedbacks (um relatório está sendo preparado especificamente sobre relações e feedbacks entre escalas).

17. A interface para a visualização de um espaço celular está muito complicada. Na verdade, o GIMS poderia simplesmente visualizar o tema da mesma maneira que ele foi visualizado no TerraView. Assim, evita-se ter que escolher cores e atributos a serem usados na visualização, isto ficaria como responsabilidade do próprio SIG. Como CellularSpaces são conectados aos temas (e não aos layers), esta estratégia seria completamente factível. Desta forma, é necessário o GIMS acessar a TerraLib diretamente para ler geometrias e atributos de um banco de dados.
18. Possibilitar a incorporação de novos modelos prontos na descrição de um modelo em específico. Mesmo sem contar com a engenharia reversa, modelos descritos no próprio GIMS poderiam ser incorporados a outros modelos. Uma investigação sobre este tópico pode gerar várias novas demandas para o GIMS.
19. Integração entre GIMS e Observer, de forma a possibilitar o usuário desenhar a interface gráfica de observação do modelo, descrevendo tanto quem envia informação aos Observers quanto configurações de cor, tamanho e dinâmica, e também os valores dos parâmetros do modelo (variáveis globais, que hoje não tem como ser representadas no GIMS). Uma boa referência é a interface do NetLogo.
20. Verificar se é realmente necessário deixar liberado para o usuário alterar o tamanho de todos os componentes. Talvez alguns componentes ficariam melhor organizados se fossem de tamanho fixo, mesmo porque hoje a maioria deles não tem qualquer conteúdo interno na representação visual.
21. Parâmetros do modelo também requerem uma atenção para o GIMS. Variáveis globais podem ser mostradas na tela e possuir valores selecionados pelo usuário antes de executar o modelo. Ver por exemplo uma tela do NetLogo, mostrada na Figura 1 com atributos inteiros (Slider), booleanos (Switch) e string (Choice). Note também nesta figura a opção de adicionar Observers (Plot e Text). Da mesma forma, será necessário adicionar estes conceitos ao GIMS.

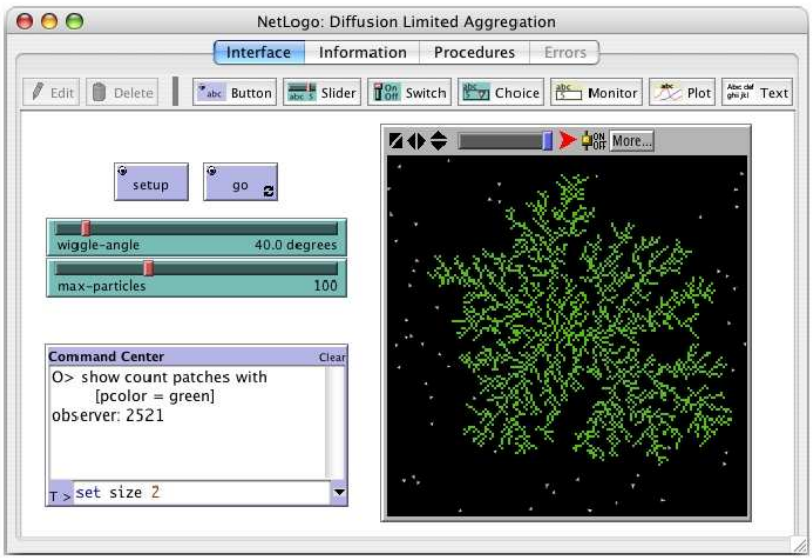

Figura 1: Descrição gráfica de parâmetros de um modelo no NetLogo.