Análise das Estratégias de Acoplamento no TerraME

Pedro Ribeiro de Andrade
12 de julho de 2011

Este relatório analisa todas as implementações de acoplamento no TerraME, levanta novos requisitos e propõe uma estrutura única na tentativa de atender a todas as implementações existentes. O principal objetivo é iniciar uma discussão sobre como integrar os esforços desenvolvidos pelos diferentes trabalhos.

Acopladores originais do TerraME (junho de 2006)

A proposta original de acoplamento entre escalas no TerraME possui as seguintes inovações:

1. **Divide o problema em três sub-acoplamentos: espacial, comportamental e temporal.**
2. **Funcionalidade para acoplar espaços celulares de mesma resolução e que possuam a mesma localização no espaço.**
3. **O acoplamento temporal é resolvido automaticamente pela hierarquia de Timers dentro dos Environments.**
4. **O acoplamento comportamental é de exclusiva responsabilidade do modelador.**
5. **Possibilita o uso da GPM para construir as relações entre células dentro de um único espaço celular (apesar de não tratar diretamente de acoplamento entre escalas, teve muita influência no desenvolvimento futuro).**

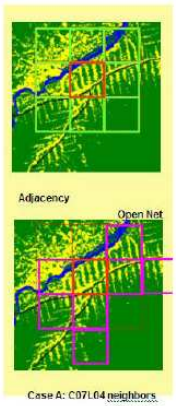
A implementação atual disponibiliza a função spatialCoupling, descrita a seguir, para o acoplamento espacial.

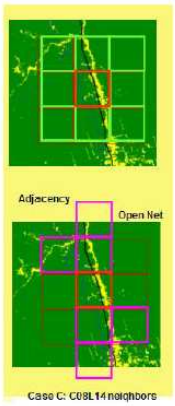
Function	Description	Example
spatialCoupling	Create an [(M+1)*2] x [(N+1)*2] (columns x rows) bidirected Neighborhood (Couclelis) between two different CellularSpaces. This function supposes that both CellularSpaces have the same resolution and extent. **M:** number of columns. **N:** number of rows. **cs1:** the first CellularSpace. **cs2:** the second CellularSpace. **filterF:** a function(Cell, Cell)→bool, where the first argument is a cell and the other is its neighbor, one from each CellularSpace. It returns whether neighbor will be included in the relation. This function is called twice for each pair of Cells, first filterF(c1, c2) and then filterF(c2, c1), wher c1 belongs to cs1 and c2 belongs to cs2. **weightF:** a function(Cell,Cell)→number, where the first argument is a cell and the second is its neighbor. It calculates the weight of the neighborhood relation. This function is also called twice for each pair of Cells. **name:** a string with the name of the neighborhood to be created.	SpatialCoupling (M, N, cs1,cs2, filterF, weigF, name)

Note que a suposição de que os espaços celulares possuem a mesma resolução e extensão para a função de acoplamento descrita acima é muito forte. O acoplamento espacial entre escalas usa a própria estrutura de vizinhança implementada no TerraME, de forma que uma relação entre células de espaços celulares distintos é considerada vizinhança da mesma forma que as relações dentro de um mesmo espaço celular. Adicionalmente, este acoplamento tem que ser necessariamente entre células (restrição da classe Neighborhood). Finalmente, a verificação referente à sobreposição espacial dos espaços celulares é de completa responsabilidade do modelador e o TerraME nada pode fazer para emitir alguma mensagem de aviso caso as relações sejam criadas com dados que não possuem uma completa sobreposição do espaço, pois o TerraME não possui acesso às geometrias dos objetos.

Locais na tese do Tiago (*INPE-14702-TDI/1227*) que citam a GPM:

- Pág. 47, na definição formal de espaço celular: "*N = {N1, ...,Nn}* is a set of GPMs – Generalized Proximity Matrix (Aguiar, Câmara et al. 2003) used to model different non-stationary and non-isotropic neighborhood relationships (Couclelis 1997). The GPM allows the use of conventional relationships, such as topological adjacency and Euclidian distance, but also relative space proximity relations (Couclelis 1997), based, for instance, on network connection relations."
  Non-isotropic está nas suposições da GPM, mas non-stationary ainda está sendo trabalhado por Raian.
- Pág. 59, em "Properties of Nested-CA model": "Space can be structurally heterogeneous in terms of proximity relations, trough the use of diverse non-isotropic and non-stationary neighborhood for different space partitions or scales. Figure 3.10 illustrates the use of generalized proximity matrix (GPM) to establish neighborhood relations considering a transportation network. Two traditional 3x3 neighborhoods (shown in the two figures on top) are compared with two GPMs that capture the topological relationships induced by the road network."
  Aqui fala-se do suporte a vizinhanças entre diferentes partições e escalas, mas não descreve que a GPM é a solução para isto. No exemplo apresentado, a GPM computa as relações entre entidades em um mesmo espaço celular. A Figura 3.10 é mostrada a seguir:





- Pág. 85, em "Database management routines": "Since the GPM neighborhoods are not yet stored in the TerraLib database, the loadNeighborhood(fileName) can be used to load a GPM neighborhood from a file whose name is received as parameter."

```
csCabecaDeBoi:load();
csCabecaDeBoi:loadNeighborhood("MooreGPM");
```

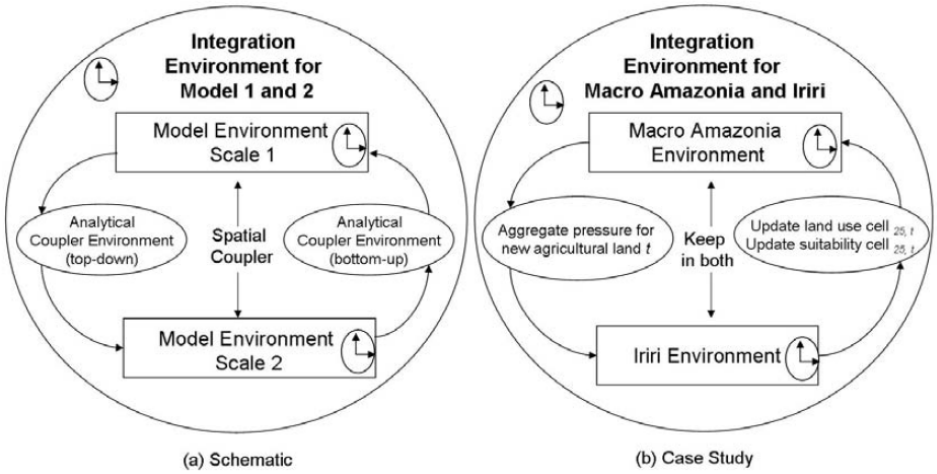
No código apresentado, a GPM conecta objetos de um único espaço celular. Os próprios formatos padrão de arquivos (gal e gwt) supõem que as relações estão sobre um mesmo conjunto de dados. Esta função foi materializada com o nome de loadGALNeighborhood.

Function	Description	Example
loadGALNeighborhood	Load a neighbourhood stored in a GAL file. **filename:** name of the file to be loaded.	cs:loadGALNeighborhood ("file.gal")

Acopladores da Tese da Eva (abril de 2009)

O acoplamento entre escalas proposto na tese da Eva baseia-se no acoplamento proposto na Tese de Tiago, apresentando as seguintes inovações:

1. **Divide o acoplamento comportamental em duas partes, top-down e bottom-up, para ajudar o modelador na descrição dos feedbacks que acontecem entre as escalas.**
2. **Possibilita a integração espacial entre espaços celulares de diferentes resoluções e extensões. Estes espaços podem estar conectados hierarquicamente ou através de uma rede.**



Assim sendo, o acoplamento da tese da Eva, exemplificado pela figura acima, possui as seguintes características:

1. Acoplamento espacial entre espaços celulares resolvido pela função spatialCoupler(cs, cs2, filename, name), e é percorrido pela função forEachRelative (novo nome basicamente por ter um índice default próprio). As estratégias

"Simple", "ChooseOne" e "KeepInBoth" são novas estratégias da GPM para espaços celulares sobrepostos.

Function	Description	Example
spatialCoupler	Load a directed Neighborhood between two different CellularSpaces from a file. **cs:** a CellularSpace (origin). **cs2:** another CellularSpace (destiny). **filename:** the file where the relations are stored. **attr:** the attribute of the Cells that contains unique identifiers used by the file to describe the relations. **name:** the name of the relation. It will be indexed as a common Neighborhood in the Cells.	SpatialCoupler(cs1, cs2, "file.gal", "object_id", "myrelation")
forEachRelative	Transverse the relatives of a given cell from another cellular space, applying a function to each of them. **cell:** an Cell. **func:** a function that takes three arguments, two cells and the weight of the relation. If some call to func returns false, forEachRelative stops and does not process any other cell.	c.value = 0 forEachRelative(c, function(c, r,w) c.value = c.value + r.value*w end)

2. Nenhuma alteração ou requisito novo para o acoplamento temporal.
3. Acoplador top-down e bottom-up são eventos a serem colocados em Timers, sem qualquer assinatura de execução pré-definida. Desta forma, a ordem de execução de cada um dos quatro componentes propostos (escala superior, escala inferior e os acopladores top-down e bottom-up) é definida pela ordem de inserção dos Environments na escala superior, de forma que quem é inserido primeiro será executado primeiro. Os trechos de código a seguir apresenta a metodologia (extraído da tese da Eva, pág. 100):

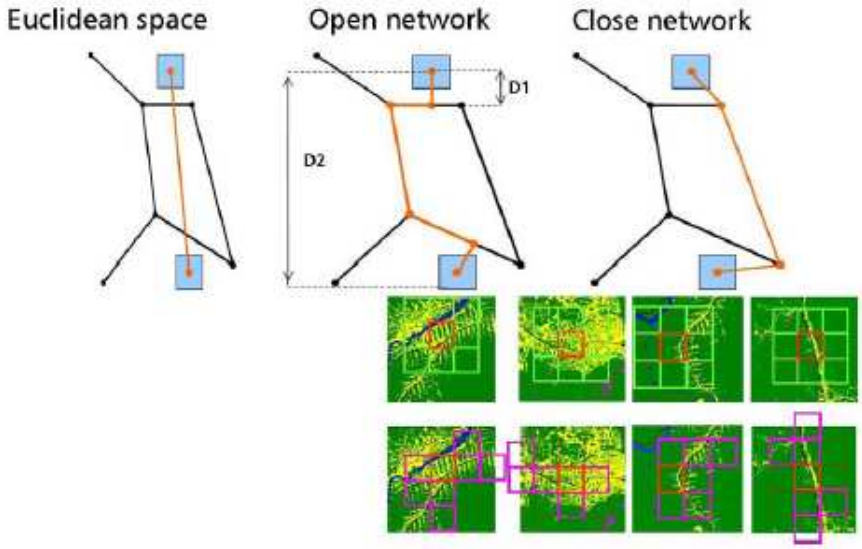
```
couplerScales = Scale{
    id = "coupler Scalers",
}

couplerScales:add(ModeloRegiona) -- add global scale
couplerScales:add(couplerTop_down) -- add top-down scale
couplerScales:add(ModeloLocal)-- add local scale
couplerScales:add(couplerBottom_Up) -- add Bottom_up scale
```

```
coupler_Top_down = Scale{
  id = "coupler Top-down",
  time = Timer{ Pair {
    Event{ time = START, period = 1, priority = 1},
    Message{ function(event)
      demandTot = coupleyByDemand(ClueScale25.csQ, SaoFelix.cs)
      aglargeracher_1.ag_demand[step] = demandTot * 0.4;
      agsmallracher_1.ag_demand[step] = demandTot * 0.6;
      return true
    end}
  }}
}
```

```
Coupler_Bottom_Up = Scale{
  id = "coupler Bottom_Up",
  time = Timer{ Pair {
    Event{ time = START, period = 1, priority = 1},
    Message{ function(event)
      aggregate(ClueScale25.csQ);
      ModeloRegional.cs:synchronize();
      return true
    end}
  }}
}
```

Locais na tese da Eva (*INPE-15795-TDI/1530*) que citam a GPM:

- Pág. 47-48, em Network-based relations: "We modify the GPM construction strategies to consider objects of different types, at different scales to support the development of multiscale landchange models (see Figure 2.8)."
  A Figura 2.8 mostra o acoplamento entre elementos de um mesmo espaço celular, e é mostrada a seguir:



- Pág. 71, em Implementation using TerraME: We added the Spatial Coupler function to TerraME. The Spatial Coupler is an extended a Generalized Proximity Matrix (GPM) that links objects of different geometries (points, lines, cells, polygons) at different scales. A GPM is generic way of expressing spatial relations between geographic objects such as cells and agents (AGUIAR et al, 2003). The original implementation of the GPM captured absolute and relative space neighbourhood relations among objects of the same type at the same scale. Moreira (2008) details how to parameterize Spatial Couplers.
  Apesar de dizer que a extensão acopla diferentes escalas tanto para agentes quanto para células, a tese mostra resultados de acoplamento apenas entre células.

Tese do Pedro (outubro de 2010) e Trabalho da Talita

O acoplamento entre escalas proposto na tese do Pedro e implementado pela Talita possui as seguintes inovações:

1. **Estende o conceito de Environment para ser usado como estrutura para acoplamento entre entidades.**
2. **Aumenta o leque de opções para a estrutura de relações, colocando os outros três tipos de relações (Agente → Célula, Agente → Agente e Célula → Agente) como podendo ser resolvidos pela GPM.**

O TerraME-ABM usa o Environment como estrutura para a união entre conjuntos de objetos espaciais (CellularSpace e/ou Society). Desta forma, apenas objetos dentro de um mesmo Environment podem ser acoplados. Exemplo:

```
env = Environment{
       id = "ENV",
       soc, -- uma sociedade
       MASCells -- um espaço celular
}
env:loadPlacement("relations.gpm")
```

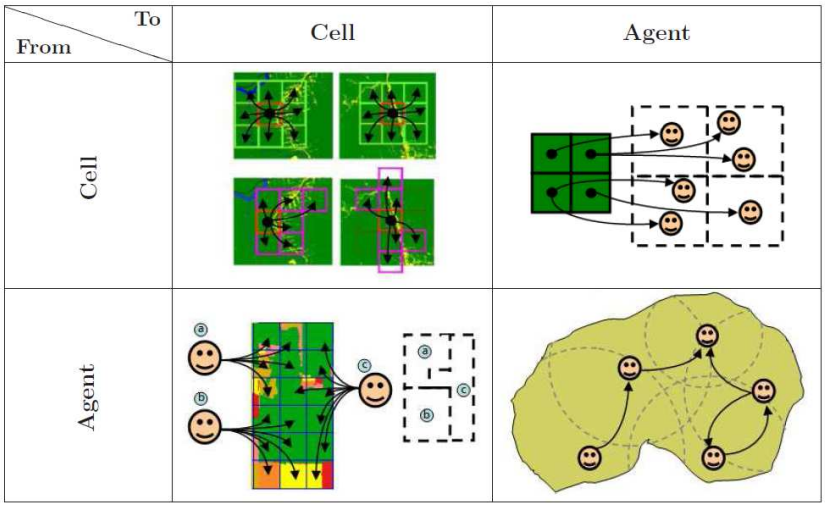
Este exemplo estabelece conexões entre entidades para relação Agente→Célula. A mesma construção poderia ser usada para fazer conexões envolvendo outros tipos de relações (Célula→Célula, Agente→Agente e Célula→Agente).

Três desvantagens surgem desta forma de integração de escalas:

- Não existe auxílio à descrição de feedbacks entre as entidades.
- Não é possível acoplar espacialmente objetos que estão em Environments diferentes, mesmo que estes Environments pertençam a uma mesma hierarquia.
- Os arquivos no formato .gal tiveram que ser alterados para suportar a identificação de quais layers de informação estes dados representam, e a partir desta diferença foi proposto o formato .gpm. Desta forma, a leitura a partir de um banco de dados seria muito mais segura, pois já conteria estas informações.

Locais na tese do Pedro (*sid.inpe.br/mtc-m19/2010/11.21.17.15-TDI*) que citam a GPM:

1. págs. 17, em Final comments, repetido na página 20: "We have as hypothesis that the GPM is a foundation for setting up the relations between the entities of an agent-based model for simulating geospatial phenomena."
   Estabelece que a GPM será usada para as quatro relações.
2. pág. 23, em GPM as basis for setting up relations: "[...] we can treat placements as multiscale relations. [...] Examples of using GPM to feed agent-based models are shown in Table 3.1. In the cell → cell case, upper neighbourhoods are created from a Euclidean space, while the lower ones use the network to calculate the nearest cells. Agent → cell and cell → agent are overlays between squared cells and agents represented by polygons and points, respectively. Agent → agent is a relation between pairs of points measured from a specific vision radius."
   Mostra como a GPM pode ser usada na criação destas relações. A Tabela 3.1 é mostrada a seguir:



3. pág. 26, em Data Interface: "Before the modeller runs the simulation, she has to prepare the database with the geospatial data and their relations, generating GPMs and selecting all the necessary data from different layers."
   Enfatiza que os dados relativos às relações têm que ser gerados antes da execução do modelo.

Implementação do acoplamento na tese da Talita:

```
-- Defines and loads celular spaces from a TerraView theme
celulas = CellularSpace{
       database = "Santarem_UTM.mdb",
       theme = "Celulas250",
}

lotes = CellularSpace{
       database = "Santarem_UTM.mdb",
       theme = "lotes"
}

comunidades = CellularSpace{
       database = " Santarem_UTM.mdb",
       theme = "comunidades",
}

-- ...

soc = Society{
       instance = AgRancher,
       database = "Santarem_UTM.mdb",
       theme = "lotes"
}

env = Environment{ soc, celulas, lotes, comunidades }

celulas:load()
lotes:load()

-- loadNeighborhood -> mesmo layer, loadCoupler -> layers diferentes
env:loadNeighborhood("celula-celula","cell-neighborhood.gal")
env:loadNeighborhood("lotes-lotes","lotes-neighborhood.gal")
env:loadCoupler("celula_lotes","celulas-lotes.gal")
```

Function	Description	Example
loadNeighborhood	Load a neighborhood stored in a GAL file to connect two CellularSpaces within an Environment. **name:** name of the relation to be created. **filename:** name of the file to be loaded.	env:loadNeighborhood ("myneigh", "file.gal")
loadCoupler	Load relations stored in a GAL file to connect a CellularSpace and a Society within an Environment. **name:** name of the relation to be created. **filename:** name of the file to be loaded.	env:loadCoupler ("myneigh2, "file2.gal")

Proposta

A proposta possui as seguintes premissas (algumas delas são premissas de todos os trabalhos acima, mas nunca foram explicitadas):

1. Modelos implementados no TerraME sob a forma de Environments deveriam poder ser usados como *read-only* (ou ao menos a ferramenta deveria se esforçar ao máximo para este fim), de forma que o modelador deveria poder usá-lo sem qualquer alteração, facilitando o reuso do código. O acesso ao código é necessário apenas para se desenvolver o acoplamento (verificar eventuais conflitos temporais, definir as relações espaciais e identificar os feedbacks entre os modelos).
2. Um mesmo CellularSpace ou Society não pode estar em dois Environments que possuam alguma relação hierárquica entre si. Se isto ocorrer, estes dois Environments deveriam ser apenas um.
3. Uma estrutura de vizinhança, para qualquer que seja o seu uso, deve armazenar entidades (células/agentes) que pertençam a um mesmo conjunto de dados. Os objetos armazenados podem pertencer ou não ao mesmo conjunto do objeto de origem, mas relações para entidades de diferentes conjuntos devem ser necessariamente armazenadas em vizinhanças distintas. Não que a ferramenta deva verificar ou proibir tal manipulação dos dados, mas é uma boa prática de programação e pode ser recomendada pelo TerraME para evitar problemas de execução do modelo.
4. No acoplamento espacial, deve existir de alguma forma uma sobreposição entre os espaços celulares, ou existir um terceiro tipo de objeto que induza esta sobreposição, apesar deste último tipo não necessariamente pertencer ao modelo. Espaços celulares que não possuam algum tipo de sobreposição não podem estar acoplados espacialmente.
5. No acoplamento de dois espaços celulares com diferentes resoluções espaciais, necessariamente a escala superior é a que possui resolução mais grossa, enquanto que a inferior é a que possui resolução mais fina.
6. A escala inferior deveria poder funcionar independentemente de a escala superior existir ou não. Assume-se que a escala inferior tem uma freqüência de execução maior ou igual à da escala superior, da mesma forma que a resolução espacial da escala inferior é tão ou mais fina do que a escala superior.
7. Qualquer conflito de execução entre duas ou mais escalas deve ser gerenciado por uma escala superior, que pode fazer ou não parte do conflito.

8. Antes da execução do modelo, supõe-se que os dados estão estáveis, o que significa que qualquer feedback necessário no tempo destes dados já teria tido algum efeito prévio. Assim, pode-se iniciar o modelo com qualquer uma das escalas, mas não com um feedback.

O objetivo principal desta proposta é aproveitar o conceito de Environment do TerraME de forma a torná-lo fundamental no desenvolvimento de modelos multi-escala. Hoje este conceito é fundamental apenas para modelos que usam autômatos celulares. A proposta é apresentada a seguir.

Espaço

- Usar o Environment como fonte de acoplamento espacial, assim como é feito no TerraME-ABM. Esta solução é genérica e serve para carregar todos os quatro tipos de relações.
- O Environment deve possuir uma função *load*, que carrega os seus espaços celulares e sociedades, bem como as relações envolvendo essas entidades, de forma recursiva, verificando todas as entidades espaciais dentro de Environments internos. Como default, assim como funciona o argumento select do CellularSpace, todas as relações armazenadas no banco de dados poderiam ser carregadas.

Function	Description	Example
load	Load all the CellularSpaces and Societies, and the relations within and between them. It executes recursively to every internal Environment. Loading relations between two different sets require that they belong to the same database, and that there is some graph connecting them in the database. The name of the relations will be the name stored in the database. If the modeler desires a different configuration, she needs to use loadRelations instead. **relations:** Contain the name of the relations to be loaded. Default is all, as long as both objects are instantiated as CellularSpaces and/or Societies.	env:load() env:load {"cell-cell", "cell-agent"}
loadRelations	Load relations stored in a database or a GAL file to connect two objects (CellularSpace/Society and/or CellularSpace/Society) within an Environment. These entities can belong directly to the environment or to another internal environment. **source:** the source where the relations are stored. It can be the name of the graph in a database, or a file name (with its path and extension), which can be a gal, gwt, or pjk. Each of these file formats require some configurations described in the Annex. **name:** name of the relation to be created. The default value is "1" when there is not any other relation in the objects; the name of the graph in the database; or the file name (without its path and extension). **bidirected:** use the connections as if they were in both directions? Default is false.	env:loadRelations ("c:/file2.gal") env:loadRelations ("mygpm", "gpm", true)

Tempo

- Acoplamento temporal atualmente implementado não está totalmente resolvido: o que fazer quando existem dois eventos a serem executados no mesmo momento e com a mesma prioridade? Note que ambos os modelos a serem acoplados deveriam ser tratados como caixas pretas, o que pode fazer com que apenas a prioridade dos eventos não seja suficiente para resolver os conflitos.
- Uma solução é o Timer poder ter, juntamente com sua lista de eventos, uma função que recebe dois eventos que irão executar no mesmo momento e com a mesma prioridade e retorna qual dos dois irá executar primeiro. Dado que várias escalas podem ter os seus próprios Timers, o Timer que será executado para resolver o conflito é o do Environment mais baixo que atende aos seguinte requisitos: (1) o Environment contem os Timers dos respectivos eventos e (2) o Timer possui alguma implementação de uma função para resolver conflitos. Note que este Timer não precisa ser do Environment de um dos Timers, mas pode ser de algum que integra os dois.
- Para esta implementação, é necessário que o Event possa saber quais entidades serão executadas, de forma possibilitar o uso de propriedades destas entidades para analisar qual das duas executará primeiro. Esta facilidade é possibilitada através de um novo parâmetro chamado target, que pode ser usado no lugar de message.
- Feedbacks devem ocorrer de forma a sincronizar o que vem acontecendo nas duas escalas. Esta proposta parte do pressuposto de que não existe uma única forma de se conceber e implementar estes feedbacks, portanto a ferramenta deve se preocupar em disponibilizar o máximo de funcionalidade possível para permitir a implementação destes feedbacks de forma simples. Exemplos: o feedback top-down pode ocorrer imediatamente após todas as ações da escala superior bem como logo antes das ações da escala inferior. A ferramenta tem que permitir estas duas estratégias, e certamente existem outras mais.
- De forma a permitir esta flexibilidade, se faz necessário existir mais um parâmetro opcional para Events: *signal*. Este parâmetro, quando usado, substitui o par (time, period) por uma condição para a ativação do mesmo. São apresentados sete sinais, mas novos podem ser propostos. Um grupo de sinais pré-definido auxilia o desenvolvimento de modelos, e alguns deles serão usados para o acoplamento comportamental descrito a seguir.
- Um último ponto para garantir esta flexibilidade é fazer com que o *message* possa receber como argumento tanto o Event que foi ativado quanto o Environment que este Event pertence, de forma a conseguir acessar as propriedades do Enviroment diretamente.

Function	Description	Example

Event	An Event represents a time instant when the simulation engine must execute some computation.
	time: the first instant of time when the event will occur (default is the current time of the Timer it will belong).
	period: the periodicity of the event (default is 1). Events are scheduled to execute indefinitely, but the results of *message* or *target* can remove it
		Event {
		time = 1985,
		period = 2,
		priority = −1,
		message = function(event)
		print(event:getTime())
		end
		}

	from the Timer.
	priority: define the priority of the event over other events. The default priority is 0 (zero). Smaller values have higher priority.
	signal: an optional string argument that replaces *start* and *period*. Instead of executing in time intervals, the event will be activated given a condition. Only a single Event can have a given signal within an Timer. These Events never have temporal conflicts with any other. The signals are:
	• **first**: executes before the first event of the Timer in a given time that has some event scheduled. It is executed when there is at least one Event scheduled for that time.
	• **last**: executes after the last event of the Timer in a given time that has some event scheduled. It is executed when there is at least one Event scheduled for that time.
	• **start**: executes when the Timer is activated, before any other event, and only once.
	• **stop:** executes right before the Timer stops reaching the final time.
	• **add:** executes every time an event is added to the Timer. It does not execute when an Event is rescheduled due to its periodicity.
	• **remove:** executes when an event is removed from the Timer. It captures both cases when it is removed manually or when the message/target returns false.
	• **vacant:** executes when the Timer does not have any event scheduled to a given time.
	message: function from where, in general, the simulation engine services are invoked. This function may have up to two arguments. The first one is an Event that was activated and the second is the Environment it belongs. If the function returns false, the Event is removed from the Timer and will not be executed again.
	target: an object to be activated. This argument is optional and it makes the event to ignore the argument *message* when used. The object passed as target must have a function called execute internally, which may have as argument an event. Another option is the target being a function itself. If the result of the function called is false then the event is removed from the Timer.
		Event {
		time = 1985,
		period = 2,
		priority = −1,
		target = agent
		}
		
		Event {
		signal = "vacant",
		message = function(ev, e)
		s="vacant time @"..e.id
		print(s)
		end
		}

Timer	A Timer is an event-based scheduler that executes and controls the simulation. It contains a set of Events and possibly a function to solve conflicts between two or more entities. It allows the model to take into consideration processes that start independently and act in different periodicities. It starts with time 0 and, once it is in a given time n, it ensures that all the events before that time were executed.	timer = Timer {
		Event { ... },
		Event { ... }
		first = function(e1, e2)
		}

Comportamento

- Como definido previamente no TerraME, a descrição de como o comportamento se materializa na integração entre escalas é de pura responsabilidade do modelador.

Estudos de Caso

Dadas definições, vou enumerar um conjunto de modelos e mostrar como a proposta atende os requisitos necessários (novas sugestões são bem-vindas):

1. Modelo com duas escalas da tese do Tiago, com pequenos produtores e grandes produtores ocupando áreas não sobrepostas no espaço, sem qualquer integração entre elas.

```
cs_small = CellularSpace{...}
cs_large = CellularSpace{...}

aut_small = Automata{...}
aut_large = Automata{...}

t_small = Timer{Event{target=aut_small, ...}, ...}
t_large = Timer{Event{target=aut_large, ...}, ...}

e_small = Environment{cs_small, aut_small, t_small}
e_large = Environment{cs_large, aut_large, t_large}

env = Envronment{e_small, e_large}

env:load() -- both cellular spaces
env:execute(2010)
```

2. Modelo simples oceano-atmosfera, composto basicamente de dois espaços celulares de mesma resolução e extensão, onde existe uma porcentagem de água que evapora no modelo de oceano que é enviada para o modelo de atmosfera. Supõe-se que a água evaporada não retorna ao oceano.

```
cs_ocean = CellularSpace{...}
cs_atmos = CellularSpace{...}

aut_ocean = Automata{...}
aut_atmos = Automata{...}

water_flux = function(event) ... end

t_ocean  = Timer{Event{target=aut_ocean, ...}, ...}
t_atmos  = Timer{Event{target=aut_atmos, ...}, ...}
t_couple = Timer{Event{signal="last", target=water_flux, ...}, ...}

e_ocean = Environment{cs_ocean, aut_ocean, t_ocean, t_couple}
e_atmos = Environment{cs_ocean, aut_atmos, t_atmos}

env = Envronment{e_ocean, e_atmos}

env:load() -- both cellular spaces and relations
env:execute(2100)
```

3. Modelo LUCC com duas escalas, uma superior com os produtores representados por suas fazendas e um inferior com um espaço celular quadrangular onde os agentes tomam as suas decisões e onde existe um autômato celular de crescimento de vegetação (trabalho da Talita).

```
farms = CellularSpace{...}

productors = Society{...}

t_produc  = Timer{Event{target=productors, ...}, ...}

env = Envronment{farms, productors t_produc}

env:load() -- cellular space, society, and relations
env:execute(2010)
```

4. Modelo com três escalas, uma escala macro e duas micro onde a alocação tem que ser dividida entre elas. Todos são modelos de alocação simples, mas com feedbacks entre eles.

```
cs_small1 = CellularSpace{...}
cs_small2 = CellularSpace{...}
cs_large  = CellularSpace{...}

function allocation_small1(event, env) ... end
function allocation_small2(event, env) ... end
function allocation_large (event, env) ... end

t_small1 = Timer{Event{target=allocation_small1, ...}, ...}
t_small2 = Timer{Event{target=allocation_small2, ...}, ...}
t_large  = Timer{Event{target=allocation_large,  ...}, ...}

env_small1 = Environment{cs_small1, t_small1}
env_small2 = Environment{cs_small2, t_small2}

env_large = Environment{cs_large, t_large, env_small1, env_small2}

function couple_top_down(_, env)
    demandTot = coupleyByDemand(env)
    env_small1.demand = demandTot * 0.4
    env_small2.demand = demandTot * 0.6
end

function couple_bottom_up()
    aggregate(env_small1)
    aggregate(env_small2)
end

t_couple = Timer{
    Event{signal="first", message=couple_bottom_up}, ...},
    Event{signal="last",  message=couple_top_down}, ...}
}

env_large:add(t_couple)
env_large:load()
env_large:execute(2010)
```