MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-14702-TDI/1227

NESTED-CA: A FOUNDATION FOR MULTISCALE MODELLING OF LAND USE AND LAND COVER CHANGE

Tiago Garcia de Senna Carneiro

Doctorate Thesis from the Post Graduation Course in Applied Computer Science, supervised by Ph.D Gilberto Câmara and Ph.D Antônio Miguel Vieira Monteiro, in June 9, 2006 .

INPE
São José dos Campos
2007

Publicado por: **esta página é responsabilidade do SID**

Instituto Nacional de Pesquisas Espaciais (INPE)
Gabinete do Diretor – (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 – CEP 12.245-970
São José dos Campos – SP – Brasil
Tel.: (012) 3945-6911
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Solicita-se intercâmbio
We ask for exchange

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-14702-TDI/1227

NESTED-CA: A FOUNDATION FOR MULTISCALE MODELLING OF LAND USE AND LAND COVER CHANGE

Tiago Garcia de Senna Carneiro

Doctorate Thesis from the Post Graduation Course in Applied Computer Science, supervised by Ph.D Gilberto Câmara and Ph.D Antônio Miguel Vieira Monteiro, in June 9, 2006 .

INPE
São José dos Campos
2007

681.3.06

Carneiro, T. G. S.
Nested-CA: a foundation for multiscale modelling of land use and land cover change / Tiago Garcia de Senna Carneiro. - São José dos Campos: INPE, 2006.
114 p. ; (INPE-14702-TDI/1227)

1. Cellular automata. 2. Automata theory. 3. Environment models. 4. Multiscale models. 5. Computer system simulation. 6. Land use. 7. Geographic Information Systems (GIS). I. Título.

Aprovado (a) pela Banca Examinadora
em cumprimento ao requisito exigido
para obtenção do Título de Doutor(a)
em **Computação Aplicada**

Dr. Reinaldo Roberto Rosa

Presidente / INPE / SJCampos – SP

Dr. Gilberto Câmara

Orientador(a) / INPE / SJCampos – SP

Dr. Antonio Miguel Vieira Monteiro

Orientador(a) / INPE / SJCampos – SP

Dr. Haroldo Fraga de Campos Velho

Membro da Banca / INPE / SJCampos – SP

Dr. Dalton de Morisson Valeriano

Membro da Banca / INPE / SJCampos – SP

Dr. Tom Veldkamp

Convidado(a) / WG / Amsterdan - NL

Dr. Waldemar Celes Filho

Convidado(a) / PUC-RIO / Rio de Janeiro - RJ

Aluno(a): **Tiago Garcia de Senna Carneiro**

São José dos Campos, 09 de junho de 2006.

“*Deus dá o frio conforme o cobertor”*.

ADONIRAN BARBOSA

A meus pais,
ALEXANDRE DE SENNA CARNEIRO e
GABI GARCIA DE SENNA CARNEIRO.

AGRADECIMENTOS

Agradeço a todos que ajudaram a concluir esse sonho.

À Fundação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, pelo auxílio financeiro de quatro anos de bolsa de doutorado.

Ao Instituto Nacional de Pesquisas Espaciais – INPE, pela oportunidade de estudos e utilização de suas instalações.

A Universidade Federal de Ouro Preto por apoiar a minha capacitação.

Aos professores do INPE pelo conhecimento compartilhado.

Aos meus orientadores Prof. Dr. Gilberto Câmara e Profº Antônio Miguel Vieira Monteiro, pelo conhecimento passado, e pela orientação e apoio na realização deste trabalho.

A todos os amigos da Divisão de Processamento de Imagem - DPI pelo saudável ambiente de trabalho, onde carinho e respeito são palavras de ordem, onde conhecimento é para ser compartilhado.

Às amigas e companheiras de pesquisa sem as quais este trabalho teria menos valor: Ana Paula Aguiar e Maria Isabel Escada.

Aos amigos com quem também tive o prazer de conviver sob o mesmo teto: Reinaldo, Gibotti, Gilberto Ribeiro, Caçapa, Caê, Henrique, e Sidão. Entre eles, aqueles que tiveram o inenarrável prazer de me adotar nos últimos anos de São José dos Campos. Pessoas com quem minha dívida será eterna: Joubert (Beto) e Flávia (Tatá).

Aos grandes amigos de aventuras e boemia sem os quais a vida perderia a graça: Brenner, Du, Liana, Borena, Felix, Aragão, Robertinha, Luciana, Rita, Malves e Brummer.

À minha amada esposa, Tenesca de Quintal Scofield Soriano, pelo amor dedicado, pela compreensão e pela infindável paciência.

À pequenina Cecília Scofield Carneiro, filhota linda que deu um novo sentido à minha vida.

A meus pais, meus avós e irmãos pelo ser humano que me tornei, por todas as graças alcançadas. Nada sou sem vocês.

ABSTRACT

This work presents the mathematical foundations of the *Nested Cellular Automata* (nested-CA) model, a model of computation for multiple scale *Land Use and Land Cover Change* studies. The main properties of nested-CA model are described and compared to the agent-based and cellular automata models of computation. The nested-CA model has been implemented in a software environment, called *TerraME* (Terra Modeling Environment), which provides a high-level modeling language for model description, a set of spatiotemporal data structures for model representation and simulation, a module for spatiotemporal data management and analysis integrated to a geographic information system, and a set of functions for model calibration and validation. We describe the main design choices involved in the development of the *TerraME* modeling environment. Its architecture is detailed and the main properties are compared with other modeling tools: *Swarm*, *STELLA*, and *GEONAMICA*. Finally, the concept of *nested-CA* and the *TerraME* architecture are demonstrated in two applications of land cover change in the Brazilian Amazon.

Nested-CA: UM FUNDAMENTO PARA A MODELAGEM DE MUDANÇAS DE USO E COBERTURA DO SOLO EM MÚLTIPLAS ESCALAS

RESUMO

Este trabalho apresenta a base matemática do modelo chamado *Autômatos Celulares Aninhados* (Nested-CA), um modelo de computação destinado ao desenvolvimento de modelos de mudança de uso e cobertura do solo em múltiplas escalas. As principais propriedades do modelo *nested-CA* são descritas e comparadas aos modelos de computação baseados em agentes e em autômatos celulares. O modelo *nested-CA* foi implementado em um ambiente computacional, chamado *TerraME*, que oferece uma linguagem de alto nível para a descrição de modelos, um conjunto de estruturas de dados espaço-temporais para a representação e simulação dos modelos, um modulo para o gerenciamento e análise de dados espaço-temporais integrado a um sistema de informações geográficas, e um conjunto de funções para calibração e validação dos modelos. As decisões de projetos envolvidas no desenvolvimento do ambiente de modelagem *TerraME* são descritas. A arquitetura do ambiente é detalhada e suas principais propriedades são comparadas com outras plataformas de modelagem: *Swarm*, *STELLA*, e *GEONAMICA*. Finalmente, o conceito de *nested-CA* e o ambiente *TerraME* são demonstrados em duas aplicações de mudança de cobertura do solo para a Amazônia brasileira.

SUMMARY

Pg.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

API	- Application Programming Interface
CA	- Cellular Automata
CBERS	- China-Brazil Earth Resource Satellite
CLUE	- Conversion of Land Use and its Effects
GIS	- Geographic Information System
GPM	- Generalized Proximity Matrix
INCRA	- Brazilian National Institute of Agrarian Reform
INPE	- Brazilian National Institute of Space Research
Layered-CA	- Layered Cellular Automata
LUCC	- Land Use and Land Cover Change
MBB	- Model Building Block
Nested-CA	- Nested Cellular Automata
SRTM	- Shuttle Radar Topographic Mission
SME	- Spatial Modeling of the Environment

CHAPTER 1

INTRODUCTION

1.1 The problem of modeling land use and land cover change

One of the most important challenges in geographical information science is to providing a computational framework for modeling environmental change. The Earth's environment is changing at an unprecedented pace. Planners and policy makers need modeling tools that are able to capture the dynamics and outcomes of human actions (Turner II, Skole et al. 1995). A particular area of interest on environmental models is the modeling of land use and land cover change (LUCC). These models aim at identifying determinant factors of land use change, envisioning which changes will happen, and assessing how choices in public policy can influence change.

An important area for LUCC studies is the process of deforestation on the Brazilian Amazonia. Some LUCC studies try to determine proximate causes and driving forces of deforestation (Pfaff 1999; Geist and Lambin 2002; Laurance, Albernaz et al. 2002; Aguiar, Kok et al. 2005). LUCC models have been applied to the region in an attempt to understand the land use change dynamics and its consequences (Laurance, Cochrane et al. 2001; Soares, Cerqueira et al. 2002; Deadman, Robinson et al. 2004; Walker, Drzyzga et al. 2004; Aguiar, Kok et al. 2005). Despite much research, there is currently no agreement as to the main causes of Amazon deforestation (Câmara, Aguiar et al. 2005). This is partly due to the lack of an established theory on human-environment interaction. On the other hand, there is a clear sense among the LUCC scientific community that human activities play a central role on the land use system (Lambin, Turner et al. 2001; Parker, Berger et al. 2001). These studies reinforce the thesis that LUCC modeling efforts should attempt to represent the multiples drivers of human-environment interaction at different spatial and temporal scales (Lambin, Geist et al. 2003; Aguiar, Kok et al. 2005; Escada, Monteiro et al. 2005).

One of the critical notions in LUCC models is the concept of *scale*. Following Gibson et al. (2000), this work uses *scale* as a generic concept that includes the *spatial*, *temporal*, or *analytical* dimensions used to measure any phenomenon. Understanding scale is important since the causes and consequences of environmental changes can be measured along multiple scales. Important aspects of scale are its *extent* and *resolution*. Extent refers to the magnitude of measurement. Resolution refers to the granularity used in the measures. In the spatial dimension of scale, *extent* is the geographical area under study and *resolution* is the geometric partition used to sample the phenomenon. In the temporal dimension of scale, *extent* is the time period considered in the analysis and *resolution* is the frequency in which changes are recorded. In the analytical dimension of scale, *extent* refers to the set of processes taken into account, and *resolution* refers to the lowest level of organization for the processes (e. g. landowner level or community level).

Earlier studies have argued that LUCC model outcomes can be strongly influenced by the chosen spatial extent and resolution (Kok and Veldkamp 2001). At different scales, changes are governed by different driving forces and different sets of processes (Turner II, Skole et al. 1995; Verburg, Schot et al. 2004). Thus, a multi-scale representation on the spatial, temporal and analytical dimensions is required for realistic environmental models, especially in LUCC studies. A single choice of extent and resolution in each of these dimensions would not be sufficient to simulate realistic geographical phenomena and reproduce expected spatial patterns.

Environmental models require proper computational frameworks. Some of the most popular computational models for LUCC are based on cellular automata (CA) models. CA models have been used for landscape and urban dynamic model development and assessment (White and Engelen 1997; Batty 1999; Almeida, Monteiro et al. 2003). These CA extensions share one limitation: the application of a single set of rules to the whole lattice. This approach has led to criticism since it cannot convey the complex motivations that drive human actions (Briassoulis 2000). In an attempt to capture these different responses, researchers have proposed the use of agent-based models for landscape and urban dynamic modeling (Parker, Berger et al. 2001). However, current

agent-based models still fall short of modeling one crucial aspect of landscape and human dynamics: *scale-dependent change*. Looking at a landscape or a city at different scales will reveal different phenomena. The cause-effect relationships that control the landscape dynamics at a smaller scale will be different from those at a larger scale (Verburg, Schot et al. 2004). For example, one of the effects of an increase of the price of grains in the international market on a developing nation varies depending on the observation scale. On a regional basis, these effects may be the construction of new roads and migration to new agricultural areas. On a local basis, they include land disputes and decisions on capital investment. Therefore, differences in scale engender differences in causative factors, which need to be translated into agent rules. Agent-based models that use a single scale will not be able to represent such scale-dependent behavior.

To overcome the shortcomings of traditional CAs and agent-based models for land use change and to allow multiscale modeling of LUCC processes, this work proposes a new type of CA: *nested cellular automata*. The purpose of a nested-CA is to allow representation of multiple scales, where each scale is associated to a specific analytical, spatial and temporal extent and resolution. Each scale is a building block of a complex LUCC model. Model building blocks are organized in a hierarchical structure, where the upper level scales provide overall control for the lower ones.

1.2 Objective of the work

The main goal of this work is to propose the concept of a Nested Cellular Automata (Nested-CA) and describe its main properties. In what follows, we provide a mathematical foundation of the concept of Nested-CA and develop a computational framework for assessment of the concept. This software environment, called Terra Modeling Environment (TerraME), is used to develop a LUCC model for the Amazon region, which explores the main properties of the Nested-CA model.

1.3 Scientific questions

This research postulates the following scientific questions:

- What are the mathematical foundations for a model of computation adequate for multiple scale LUCC studies?
- What is the best architecture for a model of computation for multi-scale LUCC studies?

1.4 Outline of the thesis

A review of the relevant literature models of computation for LUCC models is presented in Chapter 2. Chapter 3 presents the formal definition of nested CA model and compares this model with other works. Chapter 4 describes a computational architecture for spatial dynamic modeling. In Chapter 5, the concept of nested-CA and the computational architecture are demonstrated in two applications of land cover change in the Brazilian Amazon.

CHAPTER 2

THEORETICAL FOUNDATION AND PREVIOUS WORK

2.1 A brief introduction to the LUCC modeling theory and practice

According to Hestenes (1987), *modeling* is the cognitive process in which the principles of one or more theories are applied to produce a model of a real phenomenon. A *phenomenon* is any concrete fact or situation of scientific interest, which can be described or explained. Any model is an outcome from the creativity of the modeler and from the knowledge she has about the observed phenomenon. During the modeling activity, the modeler will always need to specify the structure (syntax) and functioning (semantics) of the idealized model. This specification can be represented on different ways. A *model* can be defined as a simplified and abstract representation of a phenomenon, based on a formal description of entities, their relations, and processes. Model *simulation* is the act of reproducing the behavior of some phenomenon in a computer environment. (Odum 1983; Briassoulis 2000; Parker, Berger et al. 2001).

Models where there is the time as an independent variable are named *dynamic models* (Odum 1983). Spatially explicit models or *spatial models* are models whose outcomes depend on the spatial position of each value on the input data or on the spatial pattern present on it (Parker, Berger et al. 2001). Usually, the outcomes of a spatial model are maps. A *spatial dynamic model* is a model that has a temporal structure, a spatial structure, and behavior rules that describe the changes of a spatial phenomenon (Smyth 1998; Couclelis 2000). A *LUCC model* is a spatial dynamical model that describes changes of land use and cover in a geographical area that result from the interaction of human with the environment.

Most LUCC models have a common functional structure (Veldkamp and Fresco 1996; White, Engelen et al. 1998; Lim, Deadman et al. 2002; Soares, Cerqueira et al. 2002; Verburg, Soepboer et al. 2002). LUCC models distinguish between the projection for the quantity of change and the projection for the location where these changes will take

place (Veldkamp and Lambin 2001). In the first stage, they answer the “when?” question, and establish its temporal extent and resolution. In the second stage, LUCC models have rules that govern the amount of change (the “how much?” question). In the next stage, the models determine where the projected change will take place (the “where?” question). On the final stage, the models apply the changes on an appropriate way, including external restrictions (the “how?” question). For example, a deforested location could never become primary forest again. At the end of the fourth stage, the models are back to the first stage until the simulation finishes.

There are two usual approaches for the projection of the amount of change (the “how much?” question). The first approach takes two maps about the phenomenon in different time instants and calculates the quantity of change (Veldkamp and Fresco 1996). A extrapolation method projects the observed trend on the future. The second approach uses a demographic or econometric model to calculate the quantity of change (White, Engelen et al. 1998).

For change allocation (the “where?” question), the most common approach is to calculate a change potential surface. Each location will have a numeric value indicating how prone this location is to change. Then the model traverses the surface in an ascending order of potential, applying the changes (White, Engelen et al. 1998). Some models use multi-linear regression for change allocation, such as the CLUE model (Veldkamp and Fresco 1996). Other approaches include a stochastic combination of diffusive and spontaneous change, such as the DINAMICA model (Soares, Cerqueira et al. 2002).

2.2 The modeling process

The spatial dynamic phenomena modeling process comprehends the phases described below, which not need to occur in the order they are presented. These phases are carried out several times, in a cyclic way as shown in figure 2.1, where each cycle leads to a more refined model.

- **Database development**: Acquisition and conversion of spatial data to feed the model. The database should include data for model calibration and model validation stored at several spatial scales. Geographical information systems (GIS) are the appropriate tool for managing and analyzing spatial data. Therefore, a software platform for environmental change modeling would be more useful if integrated with a GIS (Wesseling, Karssenberg et al. 1996), which provides services for data storage, aggregation, allocation and recovery at different scales.

- **Model development**: in this stage the user defines the entities that will be part of the model and the rules that will govern its dynamics. She needs to choose the scales in which the experiments will be conducted and the appropriate representations for the spatial, temporal and analytical dimensions of each scale. It is also important to model the interactions among the scales and among the model entities. Hence, a software framework toward environmental change modeling should provide services for the specification of: (a) what database data will be used as input data for the model; (b) what will be the output data and where they will be stored; (c) the time instants in which the simulation outcomes will be saved or visualized; (d) the scales considered on the model; (e) for each scale: (e.1) the entities that will be part of the model and the rules used to simulate their behaviors; (e.2) the interactions or feedbacks among entities; (e.3) the temporal order in which the entities will be simulated; (e.4) what will be the local properties or constraints in each space location; (e.5) the way the entities, possibly, traverse the spatial structure; and (f) interactions or feedbacks among different scales.

- **Model calibration, verification and validation:** after model development, the model needs to be verified to check if its implementation corresponds to idealized model. After that, it is important to calibrate the model. Calibration requires adjustment to available data. Then, the model needs to be validated by evaluating its behavior and outcomes when a different dataset is used to feed it. There are methods for calibrating and validating spatially explicit models on the

literature (Costanza 1989; Pontius 2000; Pontius 2002; Pontius, Huffaker et al. 2004). Thus, a tool for spatial dynamic model development should provide automatic methods for model calibration, verification and validation (Veldkamp and Lambin 2001).

- **Model execution and visualization; and report analysis**: in this phase the model is executed, generating summary reports and spatiotemporal data which register the model dynamics. Any modeling tool should provide services to allow the modeler to specify the report contents. It is also important that services for visualization and analysis are supplied.

- **Scenario projections:** in this stage the modeler tests hypotheses about the modeled phenomenon and try to answer "what if" question about the future in an attempt to aid the decision-making processes. Therefore, an environmental change modeling tool should offer services for the scenario development and hypotheses evaluation.

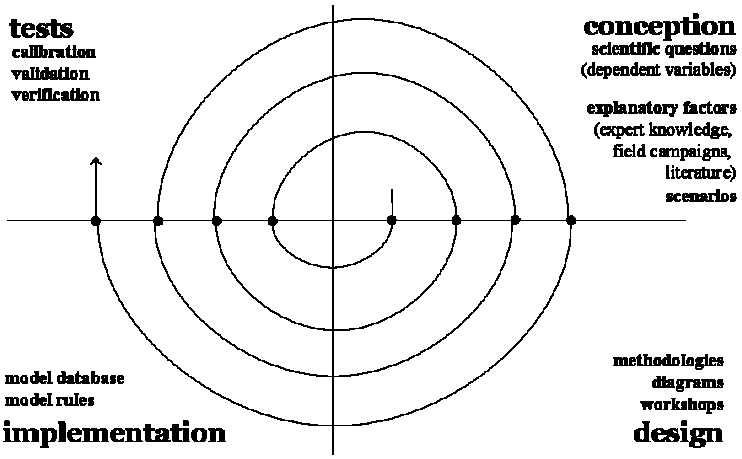


Figure 2.1 – Cyclical model development process

2.3 The role of scale in LUCC modeling

In this subsection, we consider the scale issues involved in the spatial, temporal, and analytical representations for LUCC models. As explained in Section 1, we follow Gibson et al. (2000) and use *scale* as a generic concept that includes the *spatial*, *temporal*, or *analytical* dimensions used to measure any phenomenon. The concept of

scale is associated to the need to construct discrete computer representations. We will argue that a multiple scale representation on all of these dimensions is required for realistic environmental change models.

2.3.1 Scale issues in the choice of spatial representation

Since locating change allows a better analysis of the underlying forces that cause it, spatially explicit modeling is necessary to understand geographical reality. Spatial explicit models require a choice of a discrete spatial representation. Each representation has an extent and a resolution. The choice of extent and resolution is crucial, since these factors condition the results of the model. Coarse resolution enables depiction of global patterns, but local variability can be obscured. Fine resolution shows local variability, at the expense of possibly introducing noisy patterns. A large extent will include various spatial patterns which result from different processes. A small extent might not include the whole spatial pattern. The models should assess the model at different spatial extent and resolution to improve her understanding of scale effects.

Environmental changes, at different scales, are often influenced by diverse socio-economic, biophysical, and proximate relationships that act as *driving forces* (Turner II, Ross et al. 1993; Turner II, Skole et al. 1995; Verburg, Schot et al. 2004). Consider two types of LUCC models for deforestation. The first operates at a regional scale, with a large extent and a coarse resolution. The second operates at a local scale, with a small extent and fine resolution. At a regional scale, available urban/rural infrastructure, road or market proximity, and annual rainfall are relevant LUCC driving forces. At a local scale, family structure, farm frontiers proximity, soil moisture, and modalities of land management seen to be more impelling driving forces.

2.3.2 Scale issues in choice of temporal representation

A second choice for environmental models is the choice of temporal representation. Each temporal representation will have its extent and resolution. The extent refers to the time period under consideration. The resolution is the minimum time period where the process is sampled. Land use changes are caused by different anthropogenic and biophysical processes which act at different temporal extents and resolutions. Changes

in political, institutional, and economic conditions can cause rapid changes in the rate or direction of land-cover change (Turner II, Skole et al. 1995). Government policies change typically in an yearly temporal resolution. Forest clearing and land abandonment are processes that depends on these conditions and can also present a climatic dependence at a higher temporal resolution (multi-decadal time span). Short-term rainfall variability may also have significant impact on interannual land cover change (Vanacker, Linderman et al. 2005).

As in the case of spatial representation, the choice of temporal representation is also a compromise. A sparse temporal resolution can result in a poor description of the dynamics of change, whereas a very detailed resolution may introduce noise in the studies. The choice of the temporal extent has to consider the persistence of the observed phenomena. For LUCC models, one of the temporal constraints is the limited availability of land cover data before the 1970s, where global remote sensing satellites became available. The other constraint is the long-term uncertainty of the models and the long-term error propagation. Some authors consider a period of 10 to 15 years for the maximum possible validity of LUCC models (Turner II, Skole et al. 1995).

One of the problems in LUCC modeling is that the processes represented in the model may have different temporal resolutions. Most of the anthropogenic processes are modeled in coarse resolutions, typically on yearly resolutions. Biophysical processes such as vegetation regrowth need detailed resolutions. A process may be represented in different temporal resolutions for distinct spatial extents. A LUCC model may use an annual temporal resolution to represent a deforestation process and a monthly resolution to represent changes in cultivated areas.

2.3.3 Scale issues in the choice of analytical representation

At each analysis scale, a different set of processes will cause changes. When modeling land use change, authors distinguish between proximate causes and underlying causes of change (Turner II, Skole et al. 1995). Proximate causes of deforestation are human activities that directly affect the environment. Underlying driving forces (or social processes) are seen to be fundamental forces that support the more obvious or proximate

causes of tropical deforestation. They can be seen as a complex of social, political, economic, technological, and cultural variables that constitute initial conditions in the human-environmental relations that are structural (or systemic) in nature (Geist and Lambin 2002).

At a local scale, people take decisions directly related to the management of land. Forest clearing or burning is an important process in conversion of forest into pasture or agriculture area. At regional scale, agriculture intensification and road construction are others examples of processes in LUCC. At a global scale, forest fragmentation can show a positive feedback with global warming (Laurance and Williamson 2001). In this perspective, the human dynamics of land-use change can be fitted from large- to small-scale processes (Turner II, Skole et al. 1995). Non-linearity, emergence and collective behavior may prevent a proper modeling of higher-level processes from the aggregation of detailed scale processes (Verburg, Schot et al. 2004). In this sense, process representation is scale-dependent.

The scale issues related to the choice of analytical representation for LUCC models include:

- Use of categorical or continuous variables to depict land-use change.
- The granularity of the actors involved. Models can depict individuals as actors, or may choose to capture change based on coarser scale processes such as agricultural intensification.
- The choice of the analytical model. When change is depicted as discrete events and the variables are categorical, the finite automata model is a suitable tool. When change is portrayed as a continuous event and the variables are continuous, hybrid automata (discussed in the next section) are a more suitable choice

Some studies compare the use of continuous and discrete variables in LUCC models (Southworth, Munroe et al. 2004; Binford and Cassidy 2005; Munroe and Calder 2005; Southworth and Binford 2005). They conclude that both approaches are complementary,

and that both are required to answer significant questions of land change (Southworth, Munroe et al. 2004; Binford and Cassidy 2005). At coarse resolution the LUCC process should be modeled by continuous variables, to avoid loss of model performance due to data aggregation. At finer resolution, discrete variables can be used, since data variability can be preserved.

2.3.4 Summary: the need for multiple scales

The spatial, temporal and analytical dimensions of scale establish requirements for the development of spatial dynamic models. The previous discussion point out that a LUCC model must be capable of handling multiple scales at each representation:

- *Spatial representation:* support the development of spatial models where spaces partitions can be modeled at different extents and resolutions, characterized by multiple and distinct proximity relations and described by specific local properties or constraints.

- *Temporal representation:* provide a continuous time base where discrete changes may occur, and distinct processes can change in a synchronous or asynchronous fashion.

- *Analytical representation:* support the development models where a space partition has several processes acting on it. Process can be represented by discrete and continuous variables and rules, which may belong to different analytical resolutions (e. g. individual behavior, collective behavior).

2.4 Models of computation for dynamic modeling

This section provides the mathematical formalization and a brief review on the main computational models which are the foundation for spatial dynamic models proposed in this work. These models are:

(a) The *finite automata* model (Minsky 1967), which is the conceptual basis for simulating discrete behavior. It allows simulation of process which behavior is neither sequential nor predetermined because it depends on external events.

(b) The *hybrid automata* model (Henzinger 1996), an extension of the finite automata model that allows simulation of continuous behavior.

(c) The *cellular automata* model (von Neumann 1966), which is used to simulate behavior in n-dimensional space.

(d) The situated agent theory (Rosenschein and Kaelbling 1995), which is used to guarantee a consistent behavior between an automata and its surrounding environment.

The CA model has been used for landscape and urban dynamic model development and assessment (White and Engelen 1997; Batty 1999; Almeida, Monteiro et al. 2003). Pedrosa et al. (2002) were the first to propose the replacement of the von Neumann CA discrete automaton by a *hybrid automaton* (Henzinger 1996) for LUCC modeling. This works extends their proposal by proposing LUCC models that combine *hybrid automata* theory with *situated agent* theory, and provide a support for multiscale modeling.

2.4.1 Finite automata

A *finite automata* or *finite state machine* is a abstract model for a real phenomenon or system and may be defined as a directed graph $G_g = (V, E_g)$, called *transition diagram*, where V is a finite set of vertices and E_g is a set of ordered vertices pairs named arcs (Hopcroft and Ullman 1979). Each graph vertex corresponds to one automaton state. If there is a transition from the state q to the state p, as a response to one input a, them in the transition diagram G_g there is an arc from the vertex q to the vertex p with label a. Each arc is associated to a transition rule which determines if the transition described by the arc will be executed. The finite automata model uses a discrete time base (Minsky 1967). The variable t which represents time is assigned to discrete values 0, ±1, ±2, ... The behavior of the automata is a linear sequence of events in time. Since the set of possible states is finite, a finite automaton is not appropriate to simulate behavior where the set of system states is potentially infinite. Figure 2.2 shows a transition diagram for a finite automaton capable to store a binary digit that was provided as input at the instant *t-1*. The symbol that triggers a transition is presented at the origin of the arcs.

The symbol at the middle of an arc represents the response of the machine at the transition time.

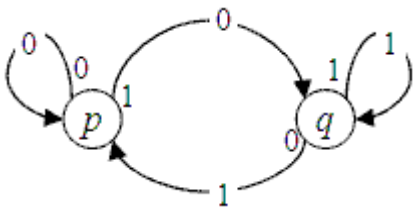


Figure 2.2 – Transition diagram for the memory machine.

Due to its simplicity, existence of an underling formal theory, and event-driven properties, the *finite automata* model (Minsky 1967) is widely used for modeling dynamical systems where the flow control is neither sequential nor predetermined because it depends on external events.

2.4.2 Hybrid automata

A hybrid automaton is an abstract model for a system which behavior has discrete and continuous components, that is, a hybrid system. A hybrid automaton consists of a finite automaton equipped with continuous variables and continuous operations over them (Henzinger 1996). A hybrid automaton extends the idea of finite automata to allow continuous change to take place between transitions. Inside each discrete state, the automaton continuous variables are allowed to change. The adoption of hybrid automata theory to LUCC models brings several benefits. One of the challenges of LUCC modeling is to combine land use change with its effects in the terrestrial and water ecosystems. For example, consider a coupled model for tropical vegetation that has a critical threshold caused by land use change. The use of a hybrid automaton would allow the modeling of the tropical vegetation system under two very different conditions. We have adapted Henzinger's hybrid automata model as a basis of LUCC models. As used in this work, a hybrid automaton H is defined by the structure (X, G, *init*, *flow*, *jump*, *method*) where:

(a) *Variables*: a finite set $X = \{x_1,...,x_n\}$ of real variables, modeled as set of points in the R^n space. The notation $X' = \{x_1',...,x_n'\}$ is used to denote the set of first

derivatives. The notation $X^* = \{x_1^*,...,x_n^*\}$ is used to denote the values of the set X at the moment of a transition between states.

(b) *Control graph*: a finite directed graph $G = (V, S)$. The vertices in V represent the discrete states of the system and are named *control modes.* The edges in S model the system discrete dynamics and are called *control switches*.

(c) *Initial condition*: The automaton H has an associated function *init*, which is the starting point of the system. It determines the initial control mode and the values of set X of model variables.

(d) *Flow conditions*: Each control mode $v \in V$ has an associated function *flow*. The flow condition *flow(v)* defines the behavior of the system inside each control mode and is generally specified as a differential equation.

(e) *Jump condition*: Each control switch $s \in S$ has an edge labeling function *jump*. The jump condition *jump(s)* is a predicate over $X \cup X^*$ and determines if a control switch will be trigged;

(f) *Method* $= \{m_1,...,m_n\}$ is a set of methods, called to obtain information about the automaton internal state, or to update the value of any variable $x \in X$.

We define a *configuration* of a hybrid automaton as a pair *(v,x)*, where $v \in V$ is the current control mode and $X^+ = \{x_1^+,...,x_n^+\}$ is the current value of its variables.

Communication between automata uses remote method invocation. Each automaton provides a set of methods that can be called by other automata. By calling methods of other automata, an automaton can obtain information about their configuration. The behavior of the automaton depends on the current control mode. This determines the flow condition that will be executed and the subset of jump conditions that may cause a transition between control modes. The hybrid automaton on the figure 2.3 models a climate variation system. The *x* variable represents the temperature. In the control mode *cooling*, the climate is becoming cooler and the temperature is declining according to the flux condition *dx/dt = -0,1x*. In the control mode *warming*, the climate is becoming

warmer and is temperature is rising according to the flux condition *dx/dt = 5-0,1x*. Initially, the temperature is 20^0 C. The jump condition *x < 19* indicates that the climate system will shift to the 'warming' mode as soon as the temperature falls below 19^0 C. The jump condition *x > 21* indicates that the climate system will shift to the 'cooling' mode as soon as the temperature is higher than 21^0 C.

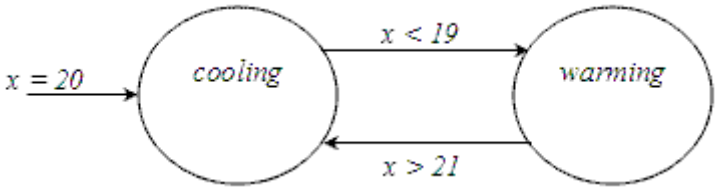
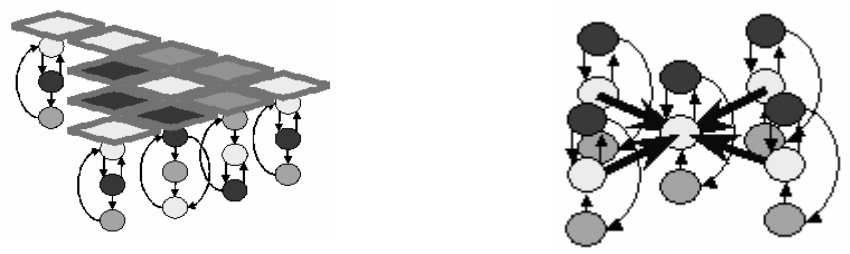


Figure 2.3 – Hybrid automata model for a climate variation system.
Source: adapted from (Henzinger 1996).

2.4.3 Cellular automata

A cellular automata (CA) as conceived by von Neumann (1966) is comprised of a finite two-dimensional lattice of squared cells, a finite automaton, and a neighborhood relationship. Each cell is occupied by a copy of the finite automaton which is connected to its four adjacent automata. As the same set of rules is present on each cell, the cellular structure is said functionally homogeneous. The von Neumann CA is isotropic and stationary. Each automaton has the same neighborhood relationship in all directions. All automata have the same configuration of neighbors. The finite automaton on each cell may be on a different initial state. Hence, one cellular space region can act on a given way and send information on a determined direction while another can behave on a different manner and send information to other direction. The *cellular automata* model (CA) is useful due to its capacity to reproduce spatial changing trough diffusion processes (Couclelis 1997; Batty 1999) and since it can simulate emergent phenomena (Wolfram 1984).

The information flow in a CA is unidirectional. When an automaton is being executed, it requests information from its neighbors. This information is combined with the internal state of the automaton to define the action it will take. Figure 2.4(a) presents the view of a portion of a CA lattice, showing the CA finite automaton on different states on each cell. Figure 2.4(b) shows the CA finite automata neighborhood relationship.



(a) (b)

Figura 2.4 – Cellular Automata: (a) same finite automaton on each cell - the cellular structure is functionally homogeneous, and (b) same neighborhood relationship on each cell - the cellular structure is isotropic and stationary.

2.4.4 Situated agents

In an attempt to capture the dynamic of phenomena whose are outcomes of several individual interactive systems act over the space, researchers have proposed the use of **agent-based models** immersed in a cellular space (Parker, Berger et al. 2001). There are different and sometimes conflicting definitions of the concept of an 'agent' (Wooldridge and Jennings 1995). This work adopts the definition provided by Russel (1995). An agent is an abstract model for an entity that is embedded in an environment. The agent is capable of sensing the environments and of acting on it. We consider that an agent has three properties: autonomy, social ability, and reactivity. To be autonomous, an agent has to control its actions and its internal state. Granting social ability to an agent requires that agents communicate. The agent should be able to perceive its environment and react accordingly. To combine the theory of agents to that of cellular automaton, each automaton has to perform as an agent. In this section, we consider an agent model (situated agents) that allows embedding agents in CAs (Rosenschein and Kaelbling 1995). A situated agent is defined by the structure $M = (S, \sum, A, \delta, \lambda, s_0)$, where:

(a) S is a set of finite internal states.

(b) $\sum$ is a set of inputs (stimulus).

(c) A is the set of outputs (actions).

(d) $\delta: S \times \sum \rightarrow S$ is a function that determines the agent's next internal state.

(e) $\lambda: S \rightarrow A$ is the function that determines the agent's next action.

(f) s_0 is the agent initial state.

An environment state ϕ can be distinguished if the modeler develops a transition function δ in such way that the agent will be in internal state s for any sequence of inputs σ^* that leads the environment to a condition ϕ from an initial condition ϕ_0. This establishes a correlation between the agent's internal state and the environment's state, and one can say that the agent is capable of recognizing the environment state.

In this model, agents are purely reactive. The environment *E* generates inputs to the agent *M*. The agent receives this input and performs some actions. These actions result in the agent reaching an internal state. One can then say that the situated agent is capable of taking decisions based on the state of the environment. The important aspect of situated automata theory is modeling systems such that, for each state of the environment *E*, there will be a corresponding state of the automaton *M*. The Figure 2.5 shows the coupling between a situated agent and its environment.

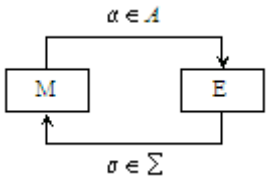


Figure 2.5 – A situated agent *M* coupled to its environment *E*.
Source: Rosenschein (1995)

2.5 Conclusion

This chapter provides a brief introduction to the basic concepts of LUCC modeling and identifies a common structure of most LUCC models. Then, it examines the process of LUCC modeling and identifies the requirements of each modeling phase. Based on these requirements, we review the concept of 'scale', considered a foundational notion. We discuss the issues related to the spatial, temporal, and behavioral representations of scale. The conclusion is that a single choice of extent and resolution in each of scale dimensions is not sufficient to simulate geographical processes and reproduce spatial patterns. The chapter also examines models of computation that will be used in the

LUCC modeling framework of the next chapters. In the next Chapter, we will show how these properties can be combined.

CHAPTER 3

THE Nested-CA MODEL

3.1 Introduction

This chapter describes the nested cellular automata (nested-CA) model and its use for LUCC modeling. The motivation for the nested-CA model is the need for adequate computational support for multiscale modeling. To understand this need, we examine the proposed extensions of the CA model on the LUCC modeling literature. Several theoretical papers have proposed CA extensions for a better representation of geographical phenomena (Couclelis 1985; Couclelis 1997; Takeyama and Couclelis 1997; Batty 1999; O'Sullivan 2001). In the specific case of LUCC modeling, recent works extend the original CA model and make it more suitable for representing the complexity of human-environment interaction (White, Engelen et al. 1998; Straatman, Hagen et al. 2001; Pedrosa, Câmara et al. 2002; Soares, Cerqueira et al. 2002; Almeida 2003).

Nevertheless, these CA extensions for LUCC modeling share one limitation: the application of a single set of rules to the whole cellular lattice. This approach has led to criticism since it cannot convey the complex motivations that drive human actions. As an alternative, researchers have proposed the use of agent-based models immersed in a cellular space (Parker, Berger et al. 2001; Batty 2005). However, current agent-based models still fall short of modeling one crucial aspect of landscape and human dynamics: *scale-dependent change*. The cause-effect relationships that control the environmental dynamics at a smaller scale will be different from those at a larger scale (Turner II, Skole et al. 1995; Verburg, Schot et al. 2004). Agent-based models that use a single scale will not be able to represent scale-dependent behavior.

As an alternative for single-scale modeling of environmental changes, some authors have proposed the *layered CA* model (Straatman, Hagen et al. 2001). The layered CA, shown in Figure 3.1, consists of two or more layers of cells. Every cell in one layer has

one parent cell in the upper layer and an arbitrary number of child cells in the lower layer. This arrangement allows the combination of models that operate in different spatial resolutions. However, the layered CA model requires a decision about the spatial stratification, where each cell is dependent on a parent cell and controls a number of child cells. The layered CA falls short of providing adequate support for multiscale modeling, since it handles only layers of fixed spatial resolutions. This approach constrains the generality of the system, since the different processes are constrained to fit the hierarchical spatial structure. In a layered CA, "*spatial structure comes before spatial processes*".

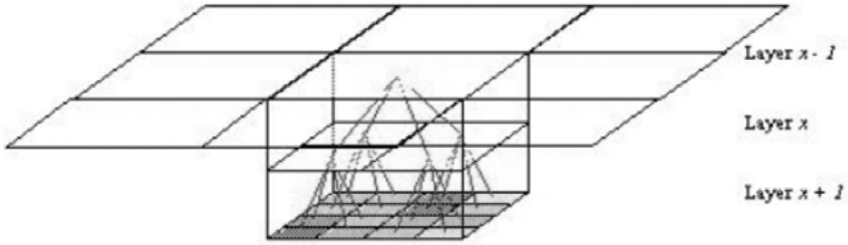


Figure 3.1 – Layered cellular automata.
Source: adapted from (Straatman et al. 2001).

To overcome the shortcomings of traditional CAs and agent-based models for environmental change modeling, we propose a new type of computational model: *nested cellular automata*, as described in the next sections.

3.2 Nested CA: a general view

The idea of a nested CA is to support multiscale LUCC modeling, where scale is defined as a particular combination of spatial, temporal, and analytical resolution and extent. A nested-CA allows scales to be defined independently and then nested to form a multiscale model. Each scale is modeled by one single nested CA which embodies all its dimensions: analytical, spatial and temporal. Each nested CA is composed of one or more cellular spaces, one or more state machines that operate in these spaces, and one or more discrete-event schedulers that control the temporal extent and resolution. Nested CAs can be embedded producing a hierarchical structure, as shown in Figure

3.2. This allows the definition of models with embedded cellular spaces, each one with its state machine changing the cell attributes at different time resolutions.

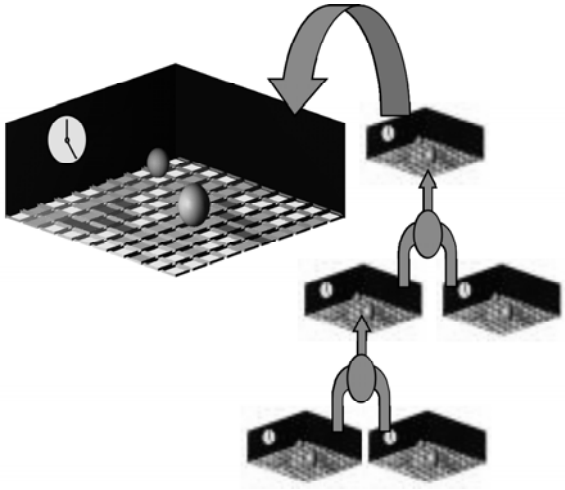


Figure 3.2 – A nested CA as a composition of nested CAs.

The nested-CA architecture is a flexible design. All possible combinations of spatial, temporal, and analytical components are allowed. One nested-CA can have two cellular spaces that share the same state machine and the same temporal resolution. Another nested-CA can have a single cellular space where different state machines operate, each with its own temporal resolution. Therefore, the concept of a nested-CA includes *spatial nesting* (one spatial extent inside another with different resolutions), *temporal nesting* (one temporal extent inside another with different temporal resolutions) and *analytical nesting* (a more general process that controls other processes of smaller granularity).

The possibility of embedding nested-CAs is beneficial for multiscale analysis, since it allows each process to be associated to the appropriate scale. The idea is that each spatial dynamical process has a suitable scale. The user should then define the spatial, temporal and analytical resolution associated to each process. Each process is then associated to a nested CA. In this way, one can develop simulations where spatial dynamic models are embedded in others. This flexibility allows diverse processes to operate in the same landscape, at different scales. In a nested CA, "*spatial processes come before spatial structures*".

3.3 Nested-CA: formal definitions

Definition 3.1 [Time Base]. The execution of a nested-CA requires a continuous time base $T \subset \boldsymbol{R}$ where discrete instantaneous events can occur.

Definition 3.2 [Event] An event is a control structure that defines when a computation must be done. Given a time base *T*, each event is defined by a structure $e = (t_o, \lambda, \rho)$, where:

- $t_o \in T$ represents the instant of time in which event must occur.
- $\lambda \in R$ is periodicity in which the event must be repeated.
- ρ is an integer that represents the event priority.

Definition 3.3 [Interface Functions]. Each nested-CA has a set of interface functions *F* that can be called, and that perform actions in the automaton. A typical set of interface functions includes loading, saving, and drawing the state of a cellular space, and to execute a specific automaton.

Definition 3.4 [Message]. The primary means of requesting actions from a nested-CA is by sending a message to it. Messages are used to invoke nested CA interface functions. Each message is associated to an event; when this event is triggered, the message is executed. Given a set of events *E* and a set of hybrid automata *H*, and a set of interface functions F associated to a nested-CA, an input message *x* is a structure $(e, h, f, \{true| false\})$, where $e \in E$, $h \in H$, and $f \in F$. The Boolean parameter $\{true| false\}$ is used to control whether the message is to be executed periodically or not.

Definition 3.5 [Message queue]. A message queue is a partially-ordered set $(Q, \preceq) = \{(e,x) \mid e \in E, m \in M\}$. Each element of the queue is a pair (*event, message*). The partial order relation $\preceq: ExE \rightarrow \{true, false\}$ is defined as

$$\preceq(e_1,e_2) = true \quad \text{if } e_1.t_o < e_2.t_o$$

$$= false \quad \text{if } e_1.t_o > e_2.t_o$$

$= true$ $\quad$ $if\ (e_1.t_o = e_2.t_o)\ and\ (e_1.\rho \preceq e_2.\rho)$

$= false$ $\quad$ $if\ (e_1.t_o = e_2.t_o)\ and\ (e_1.\rho > e_2.\rho)$

Definition 3.6 [Discrete-event scheduler]. A *discrete-event scheduler* determines when the event will be sent as input to the associated cellular automata. Given a message queue $(Q, \preceq)$, a discrete-event scheduler d is a structure (t, t_r, Q), where $t \in T$ is the scheduler internal timer that controls the simulation time. The time reference $t_r \in T$ is the time of the first event in the queue $(Q, \preceq)$. When the scheduler is executed, it removes the pair event-message (e, m) which is at the head of its queue, updates its internal timer to *event* time $(t = e.t_o)$, and executes the message m. If the Boolean parameter of message m is *true*, the discrete-event scheduler reinserts this pair event-message on its queue, according to the event's periodicity. In this case, it updates the event's time ($e.t_o = e.t_o + e.\lambda$).

Definition 3.7 [Cellular Space]. The nested CA cellular space is a set of cells defined by the structure (S, A, N, I, R), where:

- $S \subseteq R^n$ is a Euclidian space which serves as support to the nested CA. The set $\boldsymbol{S}$ is partitioned into subsets $\boldsymbol{S} = \{S_1, ..., S_n \mid S_i \cap S_j = \varnothing, \forall i \neq j, \cup S_i = S\}$.

- $A = \{A_1, ..., A_n\}$ is the set of domains of cell attributes, and where a_i is a possible value of the attribute A_i (i.e., $a_i \in A_i$).

- $N = \{N_1, ..., N_n\}$ 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.

- $I = \{(I_1, \preceq), (I_2, \preceq), ..., (I_n, \preceq)\}$ is a set of domains of indexes where each $(I_i, \preceq)$ is a partially ordered set of values used to index cellular space cells.

- $R = \{R_1, R_2, ..., R_n\}$ is a set of spatial iterators defined as functions of form $R_j:(I_i, \preceq) \rightarrow S$ which assigns a cell from the geometrical support S to each index from $(I_i, \preceq)$. Spatial iterators are useful to reproduce the spatial patterns of change since they permit easy definition of trajectories that can be used by automata to traverse the space applying their rules. For instance, the distance to urban center cell attribute can be sort in an ascendant order to form a index set $(I_i, \preceq)$ that, when traversed, allows an urban growth model to expand the urban area from the city frontier.

Definition 3.8 [Nested CA]. A nested CA is a structure of the form $N = (T, H, F, E, M, D, t_r, C, J)$, where:

- T is a time base.
- $H = \{h_1,...,h_n\}$ is a set of hybrid automata.
- F is a set of interface functions.
- E is a set of discrete instantaneous events.
- M is a set of messages.
- $(D,\preceq)$ is partially-ordered set of discrete event schedulers, where each scheduler contains a message queue $(Q, \preceq)$. The schedulers are ordered by their time references:

$$\preceq(d_1,d_2) = true \quad if\ d_1.t_r \preceq d_2.t_r$$

$$= false \quad if\ d_1.t_o > d_2.t_r$$

- t_r is a time reference for the nested-CA. The time reference $t_r \in T$ is the time of the first event scheduler in $(D,\preceq)$.

- $C = \{c_1,...,c_n\}$ is a set of cellular spaces. Although it is possible to define any arbitrary structure for model the space, in this thesis we assume a regular grid structure for simplicity.

- $(J,\leq)$ is partially-ordered set of nested-CAs $\{j_1,...,j_n\}$. The nested-CAs are ordered by their time references:

$$\leq(j_1,j_2) = true \qquad if\ j_1.t_r \leq j_2.t_r$$

$$= false \qquad if\ j_1.t_o > j_2.t_r$$

3.4 The Nested-CA models of computation

A nested-CA provides two different models of computation for spatial process modeling and simulation. The *global automaton* model allows the development of models based on the *agent* approach. A global automaton is an individual that traverses the cellular space, one cell after another, evaluating its rules at each position. Changes occur sequentially in the cellular space. The global automaton has a single internal state. The *local automaton* model allows the development of models based on the *cellular automata* approach. Each cell has its own internal state. At each iteration, each cell changes its state independently, based on a common set of rules. Changes occur in parallel in the cellular space, and all locations may change simultaneously (see Figure 3.3).

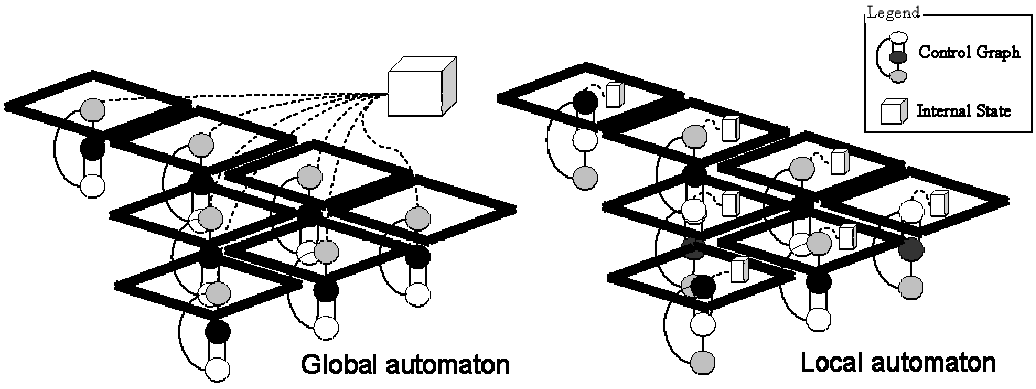


Figure 3.3 – The internal state of a hybrid automaton keeps track of the current active control mode and of the continuous variables values.

Another way to compare the *global automaton* and *local automaton* models is shown in Figure 3.4. The sequence of changes in the model state during the simulation of a *global automaton* is shown in Figure 3.4(a). Changes in the space follow the trajectory of the process. The automaton state is represented by a global value shared in all cells. When the automaton is executed in a *cell*, changes in its internal state are perceived instantaneously in all cells. In the *local automaton* model, the processes are autonomous. At each location, they can be in different state and exhibit a different behavior. Figure 3.4(b) shows the sequence of changes in the model state during the simulation of a *local automaton*. When the automaton is executed in a cell, only the copy of the automaton internal state in that cell will be updated. Changes in the internal state are perceived locally.

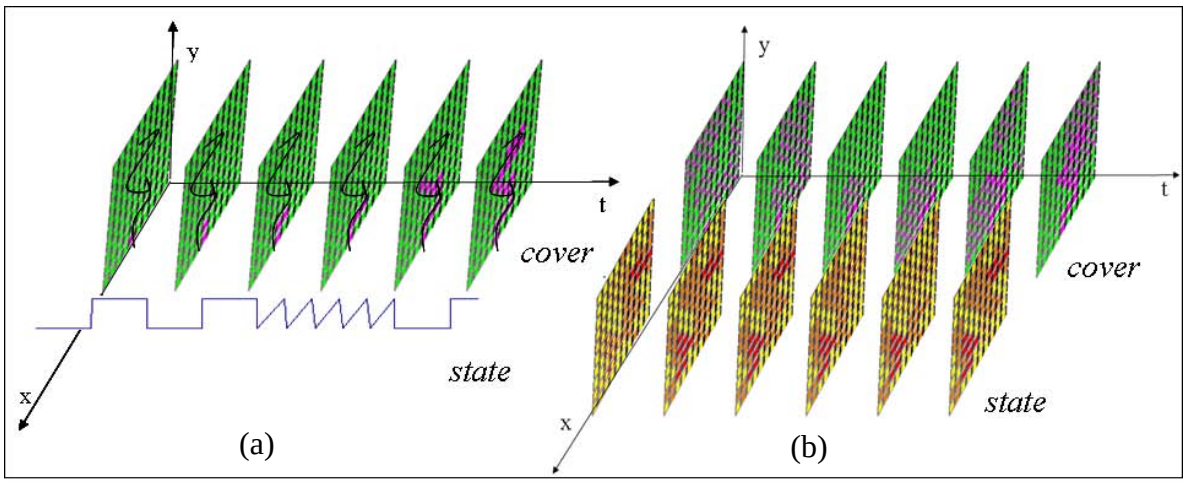


Figure 3.4 – Changes in the cellular space and in the automaton internal state in: (a) A *global automaton* model; (b) A *local automaton* model.

Definition 3.9 [Global Automaton]. The global automaton is an abstract model for a system whose internal state does not depend on a determined location. A global automaton is a hybrid automaton h_g = (*X*, *G*, *init*, *flow*, *jump*, *method*), as defined in section 2.4.2. Recalling the definition of hybrid automaton, the graph *G* has a set of vertices *V* (*control modes)* and a set of edges *S (control switches)*. In a global automaton, the *flow* conditions and *jump* conditions are defined as:

- *flow(v)* is a node labeling function that assigns to each *control mode (vertex)* $v \in V$ a function $f:V\times B \rightarrow B$ that describes the automaton continuous behavior, where $B = E\times I\times A\times N\times X$. *E* is the set of events associated to the nested-CA. *I* is a set of

spatial iterators. *A* is the set of domains of cell attributes, and *N* is the set of proximity matrixes. *X* is the set of the automaton continuous variables. The function *f* is defined by $b^t = f(v^{t-1}, b^{t-1})$, where b^t is a value in *B* at time *t*, b^{t-1} is a value in *B* at time *t-1*, and v^{t-1} is the automaton *control mode* at time *t-1*. A *flow condition* selects an action based on the current automaton *control mode*, on the *event* that triggers it, on the index (location) of a cell where it is being evaluated, on the values of the cell attributes, on the cell neighborhood, and on the *continuous variables* of the automaton.

- *jump(s) is* a edge labeling function, named *jump condition*. It assigns to each *control switch* (edge) in *S* a function $j: V \times B \rightarrow V$ that determines if the *control switch* will be triggered and the automaton is transferred to a new discrete state. *B* and V are as defined above. The function *j* is defined by $v^t = f(v^{t-1}, b^{t-1})$, where $b^{t-1} \in B$, and v^t and v^{t-1} are the automaton *control modes* at times *t* and *t-1*.

Definition 3.10 [Local Automaton]. The local automaton is an abstract model for a system which internal state depends on the location in which it is evaluated. A local automaton is a hybrid automaton $h_l = (X^c, G, init, inv, flow, jump, method)$, where:

- *G, init, flow, jump,* and *method* are defined as above.
- $X^c = \{(c_i, X_i) \mid c_i$ *is the i-th cell from the nested CA cellular space,* $X_i = \{x_1^i, ..., x_n^i\}$ is the finite set of real variables associated to c_i *}.*

3.5 The semantics of the Nested-CA model: a situated hybrid automaton

The preceding sections describe the structural aspects of the nested-CA model. The above definitions indicate that the nested-CA model has a rich semantics, which combines the idea of hybrid automata, multiscale models, and global and local models of computation. This section discusses how the nested-CA structure works.

The first important issue is the *situated semantics* of the nested-CA. The original definition of a hybrid automaton (Henzinger 1996) and its adaptation to LUCC modeling (Pedrosa, Câmara et al. 2002) do not describe how to guarantee a consistent

behavior between an automata and its surrounding environment. The nested-CA model therefore requires the combination of hybrid automata theory with situated agent theory (Rosenschein and Kaelbling 1995). When an automaton is simulated, all the *jump conditions* of the current *control mode* are evaluated, before any *flow conditions* of this *control mode* are executed. If a transition to another *control mode* occurs, all *jump conditions* of the new *control mode* are checked. This process goes on until a *control mode* that reflects the nested-CA state is reached. When the correct *control model* is reached, its *flow* conditions are executed.

The second issue is the *semantics of scheduling*. Events must occur in a chronological order from a given initial time t_0. When a pair (*event*, *message*) is removed from or inserted into a discrete-event scheduler queue, this scheduler changes its position in the partially-ordered set of schedulers $(D,\leq)$ associated to the nested-CA. This leads to a reorganization of the partially-ordered set $(J,\leq)$ of internal nested-CAs.

The third issue is the *semantics of synchronization*. A nested-CA has one or more automata, which share a common set of model variables. Each variable has a timestamp registering the instant of its last updated. The cell space attributes are example of variables shared by all automata. To guarantee the consistency of its models of computation, all automata must agree with the order the changes have occurred. For the same input data, any computation on shared variables should always result in the same output value, and all automata should agree on this value. However, if two automata attempt to update the cellular space attributes simultaneously, a race condition occurs and the cell values might become unsynchronized. To solve this problem, the nested-CA model requires the modeler to explicitly synchronize the shared variables. At any point of the simulation, the modeler can call the interface function *synchronize*. It is used to synchronize either a cell (all attributes values will receive the same timestamp), a cellular space (all cells have will receive the same timestamp), or a nested-CA (all cellular spaces and internal nested-CA will receive the same timestamp). In this way, the modeler controls the synchronization and allows changes to be propagated. There are no hidden assumptions on the order that simultaneous automata will update the shared variables.

The nested-CA concurrency model guarantees that all automata will record the changes in the same chronological order. It provides four synchronization schemes, as shown in Table 3.1, where the method *execute* is a nested-CA *interface function* that executes an automaton:

- *Sequential in space and time:* The automata act sequentially in space (they are *global automata*) and in time (the outcomes of the first automaton actions are input for the second automaton rules). After each automaton is executed, it must synchronize its results with the shared variables.

- *Sequential in space, parallel in time*: The automata execution is sequential in space (*global automata*) and parallel in time (changes occur simultaneously). The shared variables are synchronized after the execution of both automata.

- *Parallel in space, sequential in time*: The automata act simultaneously on several space locations (*local automata*) and are serialized in time. The shared variables are synchronized after the execution of each automaton.

- *Parallel in space, parallel in time*: The automata act simultaneously on several space locations (*local automata*) and changes occur simultaneously. The shared variables are synchronized after the execution of both automata.

TABLE 3.1 – The Nested-CA Synchronization Schemes.

	Sequential in Time	Parallel in Time

Sequential in Space	execute(globalAutomaton1);
	synchronize( );
	execute(globalAutomaton2);
	synchronize( );
		execute(globalAutomaton1);
		execute(globalAutomaton2);
		synchronize( );

Parallel in Space	execute(localAutomaton1);
	synchronize( );
	execute(localAutomaton2);
	synchronize( );
		execute(localAutomaton1);
		execute(localAutomaton2);
		synchronize( );

The last issue is the *semantics of communication*. Communication between automata uses remote method invocation. By calling these methods, an automaton can obtain information about the current *control mode* and *continuous variables* of the others. Since automata are autonomous, one can never set the *control mode* of another. However, it can use the methods to update the values of the continuous variables of another.

3.6 Modeling using a nested-CA: an example

When using a nested-CA to model a specific problem, the modeler should follow a general guidance:

- Identify processes that have *global* rules and *global* behavior as *agents*. Model those as global automata.
- Identify processes whose states are location-dependent rules. Model those as local automata.
- For each automaton, define the discrete behavior using jump conditions and the different types of continuous behavior using flow conditions.

- For each automaton, define the methods that will be used for intercommunication.
- Create one of more nested-CAs, each with its spatiotemporal resolution and extent, defining a cellular space (spatial resolution and spatial extent) and an event scheduler.
- Associate each automaton to a nested-CA.
- Embed one nested-CA inside another, if required for multiscale modeling.

3.6.1 A hydrologic balance spatial dynamic model

As an example a nested-CA, consider a very simplified hydrological balance process. The idea is to simulate rain drainage in a terrain. Only superficial drainage is considered. The analytical dimension of the model is composed of two automata: a *global automaton* that simulates the rain and a *local automaton* that simulates the water balance process, as shown in Figure 3.5. The rain automaton has one *control mode*, with one *flow* condition: a constant rain. The water balance automaton has two *control modes*: *dry* and *wet*. In the *dry* control mode, there are no *flow conditions*. In the *wet* control mode, the amount of water retained is made equal to the infiltration capacity, and them the surplus water is sent downhill. In this example, there is no intercommunication between the automata.

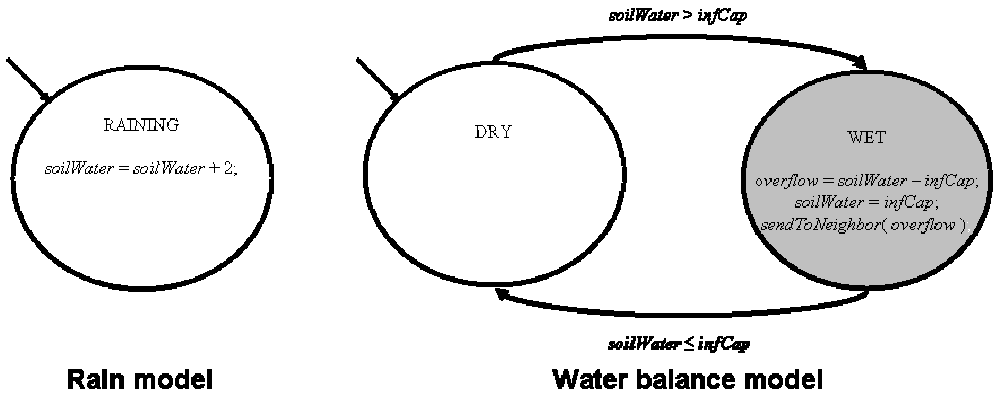


Figure 3.5 – The rain model (left) and the water balance model (right).

The next step is to determine the extent and resolution of the *cellular space* and *discrete-event scheduler* used in the model. Based on SRTM (*Shuttle Radar Topographic Mission)* data, we model the space by a 90 x 90 meter regular cellular space, as shown in Figure 3.6. The cell space uses a 3x3 neighborhood, which is composed by the eight immediately adjacent cells of a certain cell. The cell attributes are: amount of water in the soil (*soilWater*), cell elevation (*altitude*), and cell infiltration capacity (*infCap*). An infiltration capacity of 0.5 mm/hour has been considered for all cells and the initial amount of water in each cell is zero. The *flow* condition of the rain automaton is a rain of 2 mm at the start of the simulation.

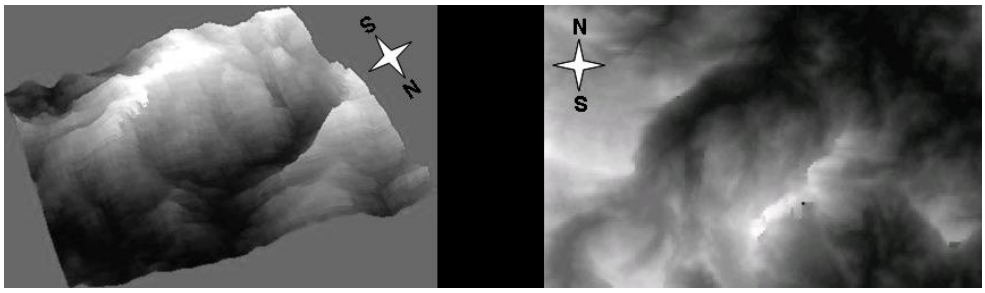


Figure 3.6 – Terrain digital model (left) based on the SRTM data (right). Light gray pixels denote higher locations while dark gray lower ones. The maximal elevation is 1550 meters and the lowest is 1100 meters.

The nested-CA has a temporal resolution of one minute. A discrete-event scheduler, with two pairs (*event, message*) is inserted into the nested CA. The event $e_1 = (0, 3600, 0)$ triggers the message $m_1(e_1, rain, \text{“}execute(rain); synchronize( );\text{”}, false)$ at the start of the simulation. The event $e_2 = (0, 1, 0)$ triggers the message $m_2(e_2, waterBalance, \text{“}execute(waterBalance); synchronize( );\text{”}, true)$ at each minute. The message m_1 activates the *rain automaton* and synchronizes the nested-CA. The event e_1 will never be reinserted on the scheduler queue. The message m_2 activates the *waterBalance* automaton and synchronizes the nested-CA. The event e_2 will be inserted into scheduler queue every minute. Figure 3.7 shows the spatial pattern of water balance process at different simulation times.

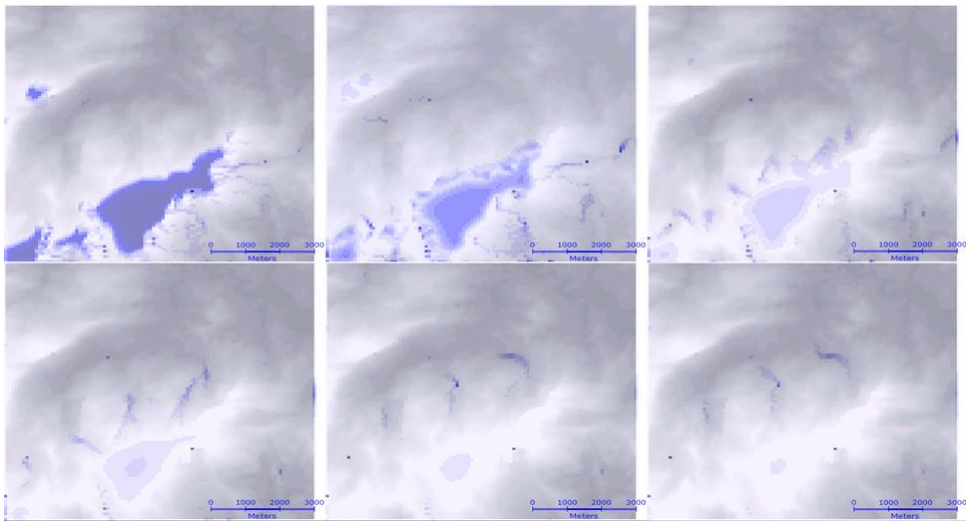
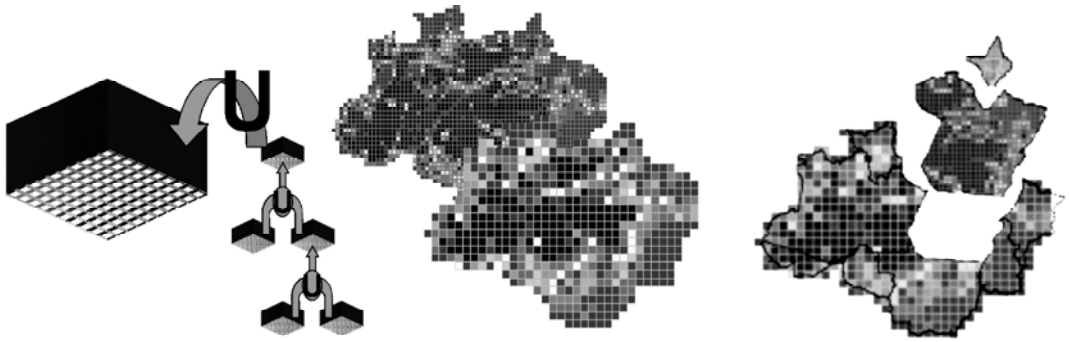


Figure 3.7 – Spatial temporal pattern of precipitation being drained: from the top left to the bottom right map.

3.7 Properties of the nested-CA model

This section considers the properties of the nested-CA model. Given the formal definitions presented above, the nested-CA model has the following properties:

- **Space can be structurally heterogeneous in terms of scale and driving forces.** The cellular space in each nested CA will determine the spatial resolution and the cell attributes perceived by all automata inside it. Multiscale models, as shown in Figure 3.8, can be constructed composing various nested CAs, which can have different spatial extents and resolutions.



(a) (b) (c)

Figure 3.8 – Nested cellular automata (a), multiple scales (b) and multiple resolutions in different space partitions (c).

- **Behaviour can be heterogeneous in space and time**. Different processes act upon different space partitions, with different time resolutions, as exemplified in Figure 3.9. In Figure 3.9.a, the ocean could be modelled as one nested CA (with specific processes, such as salinity variation or oil spill spreading), and the land as a different nested CA, also with land specific processes (such as deforestation or natural vegetation growth). The total scene could be modelled as a third nested CA with common process (such as climate or weather) that includes the two other nested-CAs as its components. In Figure 3.9.b, in a typical Amazon area of intense deforestation, one can notice different actors and processes that could be modelled in different ways. In more consolidated area, an intensification process is beginning to happen, with more capitalized actors. In a recent deforestation area, the actors are non-capitalized small farmers, living on subsistence agriculture.

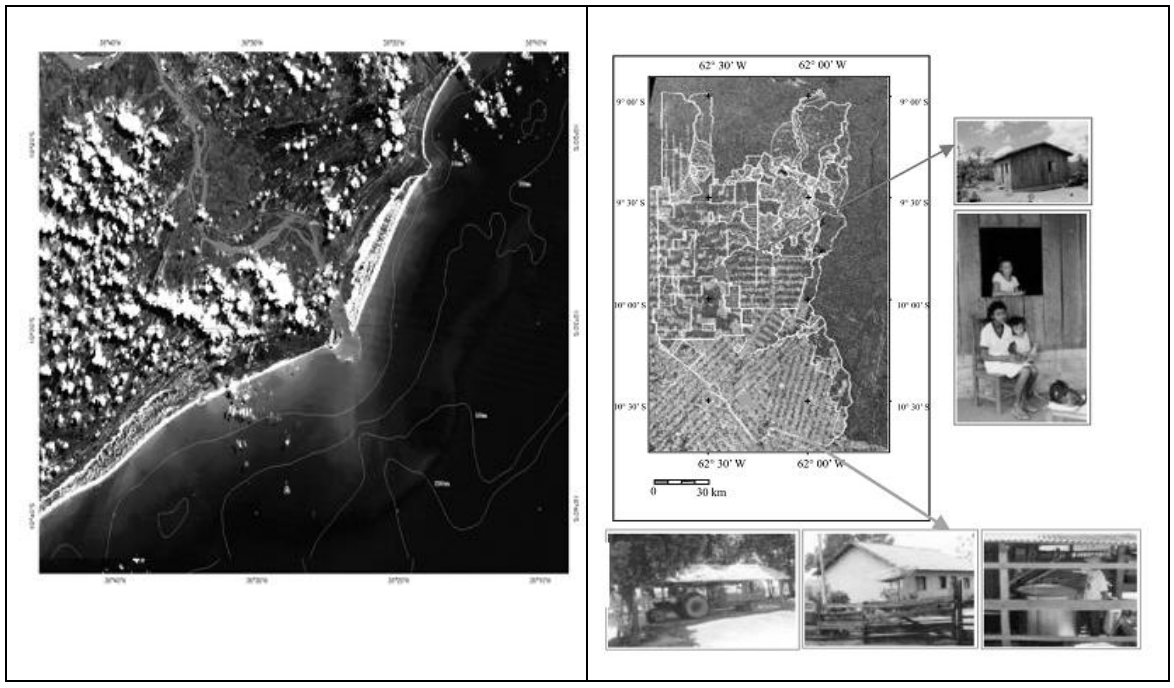


Figure 3.9 – Different processes act on distinct space partitions: (a) coastal area, (b) settlements area in Rondônia, Brazil.
Source: adapted from (Escada 2003).

- **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.

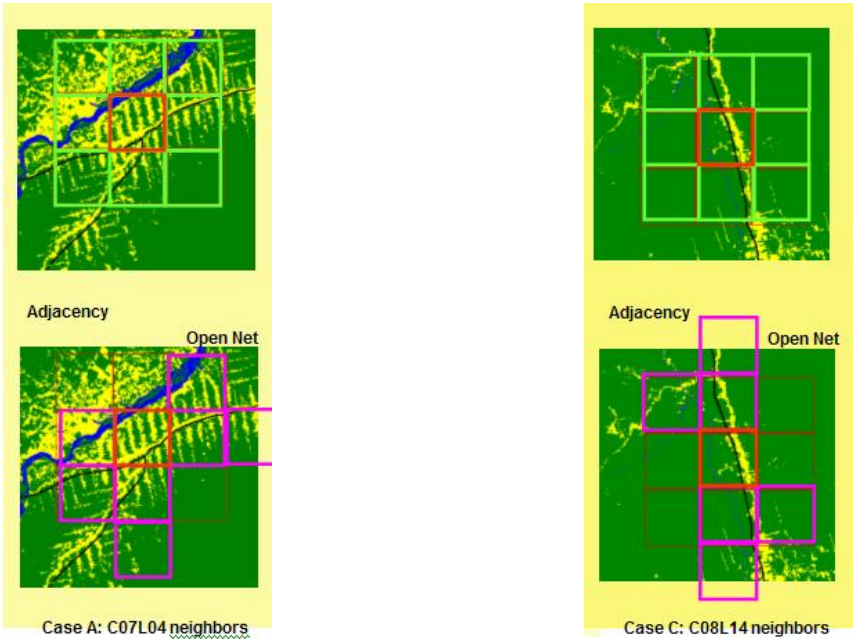


Figure 3.10 – Generalized Proximity Matrix for modeling non-isotropic processes: Amazon deforestation processes and roads (a), Moore neighborhood (b) and road geometry based neighborhood (c) for the red central cell. Source: adapted from (Aguiar et al. 2003).

- **Spatial dynamic processes can be asynchronous.** Since each nested-CA can be independently synchronized, the associated automata can operate at different spatial partitions at distinct temporal frequencies.

3.8 Comparison with previous works

The nested-CA is a model of computation where all scale dimensions (spatial, temporal and analytical) can be modeled independently. Multiscale models can be structured by the composition of several nested-CA. In this section, we provide a comparison between the nested-CA model, and other models: the layered CA model (Straatman, Hagen et al. 2001) and agent-based models (Parker, Berger et al. 2001).

The layered CA model provides a structure where the spatial dimension of the scale concept can be modeled in diverse extents and resolutions. It is not clear how the various extents and resolutions of the temporal and analytical dimensions are related to the spatial ones to represent each scale. This architectural approach does not provide a clear and direct answer to simple questions involved in multiple scale modeling. For example: *If a cellular layer is removed from the model, what analytical models should be removed? How to represent processes that are confined in different space partitions that have the same resolution (e.g. salinity variation in the ocean and deforestation in the land)? How to represent processes that are driven by different driving forces in distinct space partitions, that is, how to represent space partitions where cells have distinct attributes?* By contrast, all of these problems can be modeled using a nested-CA.

The agent-based model is not spatially explicit. It does not provide high level abstractions for representing spatial processes, scale dependent behavior, or spatial patterns of change. Therefore, it does not provide suitable abstractions for multiple scale modeling as discussed in this work. A nested-CA can simulate an agent-based model by using global automata. Using a nested-CA with local automata provides a flexibility which is not possible in agent-based model.

The nested CA architecture allows different scales or space partitions to be occupied by discrete or continuous process, and non-isotropic and non-stationary neighborhood relations. Neither the layered CA model nor the agent-based models provide devices to represent continuous process and such proximity relations. The concept of spatial

iterator allows a nested-CA to reproduce spatial patterns of change. Neither the layered CA model nor the agent-based models provides abstraction for this purpose.

3.9 Conclusion

In this Chapter, we identified the main requirements of a computational model for multiscale environmental change modeling. We proposed a new model of computation, called **Nested Cellular Automata** (nested-CA) that satisfies these requirements. The nested-CA structure allows the development of complex dynamic spatial models from hierarchically organized simple ones. It is possible to build models in which different geographical space partitions have several actors and processes. A nested-CA simulates discrete or continuous behavior. Neighborhood relations may be defined in non-isotropic and non-stationary topologies. Spatial iterators reproduce how different spatial patterns change. The nested-CA model also supports the development of LUCC models based on the traditional CA or agent-based approaches. We argue that:

- The nested CA is a model of computation suitable to support multiple scale environmental change model development and assessment.

- Both agent-based and CA-based models for environmental change simulation can be expressed as specialization of the nested-CA model.

The nested CA model has been implemented in a software platform named *Terra Modeling Environment – TerraME*, which will be detailed in the next Chapter. In Chapter 6, some multiple scale land use and land cover change models that have been developed through the use of the *TerraME Framework* will be discussed.

CHAPTER 4

TerraME: A LUCC MODELING FRAMEWORK

4.1 Introduction

In this Chapter, we present the design and implementation issues involved in the development of a software platform for nested-CAs. This software platform uses the *TerraLib* spatial library developed by INPE (Câmara, Souza et al. 2000) and is called *TerraME* (*Terra Modeling Environment*). The *TerraME* environment implements the nested-CA model and services for spatiotemporal data analysis and management, model development, simulation, and assessment. This chapter discusses critical design decisions, system architecture, and implementation strategies.

Several modeling environments have been developed or used for LUCC modeling, including *SME* (Maxwell and Costanza 1995), *Swarm* (Minar, Burkhart et al. 1996), and *Kenge* (Box 2002). The *SME* framework (Maxwell and Costanza 1995) integrates a cellular space with GIS systems and embeds a *STELLA* model in each cell. The *STELLA* modeling tool (Roberts, Anderson et al. 1983) is an implementation of the dynamic system description language proposed in (Forrester 1968). This language uses flow diagrams, feedbacks loops, and differential equations to describe continuous systems.

Swarm is a library of classes and objects for the development of multi-agent simulations, where several discrete-event schedulers can be declared to allow agents to act on different time resolutions (Minar, Burkhart et al. 1996). It does not provide spatial abstractions. The *Kenge* toolkit implements a GIS integrated cellular space for the *Swarm* platform (Box 2002). Several others agent-based platforms have been used for LUCC modeling and are compared in (Parker, Berger et al. 2001). The most powerful property of *STELLA* and *Swarm* platforms is the existence of abstractions to organize models in a hierarchical way, allowing complex models to be developed from the composition of simpler models in a “black box” fashion.

Despite the positive aspects of modeling environments such as *SME* (that uses *STELLA*) and *Kenge* (that uses *Swarm*), they do not provide support for the full set of requirements for multiscale modeling, as discussed in the previous Chapter.

4.2 Design choices

This section describes the design decisions for building a computational environment that implements the nested-CA architecture. This environment requires five main services: model description, model representation, simulation engine, model assessment, and spatiotemporal data management.

Supporting model description requires an expressive modeling language to allow quick prototyping. Although a graphic representation is useful to depict parts of a model, the rules of any model will always need to be defined by the modeler. The language should have high-level constructs to allow easy model understanding. It also should be extensible to include new data types. To support this needs, we chose LUA as the basis for the model programming language. LUA is an extensible programming language especially designed for extending applications (Ierusalimschy, Figueiredo et al. 1996). LUA is an open source project, and the language is very simple and expressive.

The use of a well known extension language avoids the costs of a new language design and interpreter development. LUA has a large amount of programmers in the game development community, an activity that has many requirements in common with simulation. Among the existing extension languages (such as Phyton, Tcl, Perl, and Visual Basic), LUA presents simpler syntax and best performance (Ierusalimschy, Figueiredo et al. 1996). A LUA plug-in for the Eclipse development environment provides syntax highlight for the programs, improving model legibility.

Model representation requires data structures that define hierarchically organized scales where the higher levels in the hierarchy provide overall control over the lower levels. The simulation engine should support concurrent programming, where geographical processes are represented by independent control flows. This requires data structures

and algorithms for scheduling, communication and synchronization of model components.

Model assessment includes calibration and validation methods that compare the agreement between two maps in several resolutions (Costanza 1989) and attempt to distinguish between errors of location (allocation) and quantity (demand) (Pontius 2002; Pontius, Huffaker et al. 2004).

To support spatiotemporal data management, the best solution is to integrate the modeling environment in a GIS. However, there should be no dependence of a specific GIS technology. To this end, a properly-designed application programming interface (API) should encapsulate all data management services. Specific versions of this API allow the environment to communicate with different spatiotemporal databases. We have chosen to implement the modeling environment using the *TerraLib* GIS library, which implements a spatiotemporal database over relational database systems (e.g. *MySQL, PostgreSQL, Access, Oracle*) (Câmara, Vinhas et al. 2001). *TerraLib* provides support for cellular spaces, whose neighborhood relations can be defined trough the use of generalized proximity matrices (Aguiar, Câmara et al. 2003).

4.3 TerraME: a general view

TerraME is a modeling environment that implements the nested-CA model to allow the development of spatially explicit LUCC models where several temporal, spatial and analytical resolutions and extents are taken in account. The *TerraME modeling language* is a LUA programming language extension. Using this language, the nested CA model can be specialized to implement models for specific cases. *TerraME* is coupled with the open source *TerraLib* GIS library (Câmara, Vinhas et al. 2001), which provides services for model input and output data storage. Data can be visualized and explored using the *TerraView* software, a viewer that demonstrates the visualization, spatiotemporal query, and spatiotemporal analysis functions of *TerraLib*. *TerraME* implements calibration and validation methods to spatially explicit dynamic models assessment (Costanza 1989) (Pontius 2002; Pontius, Huffaker et al. 2004).

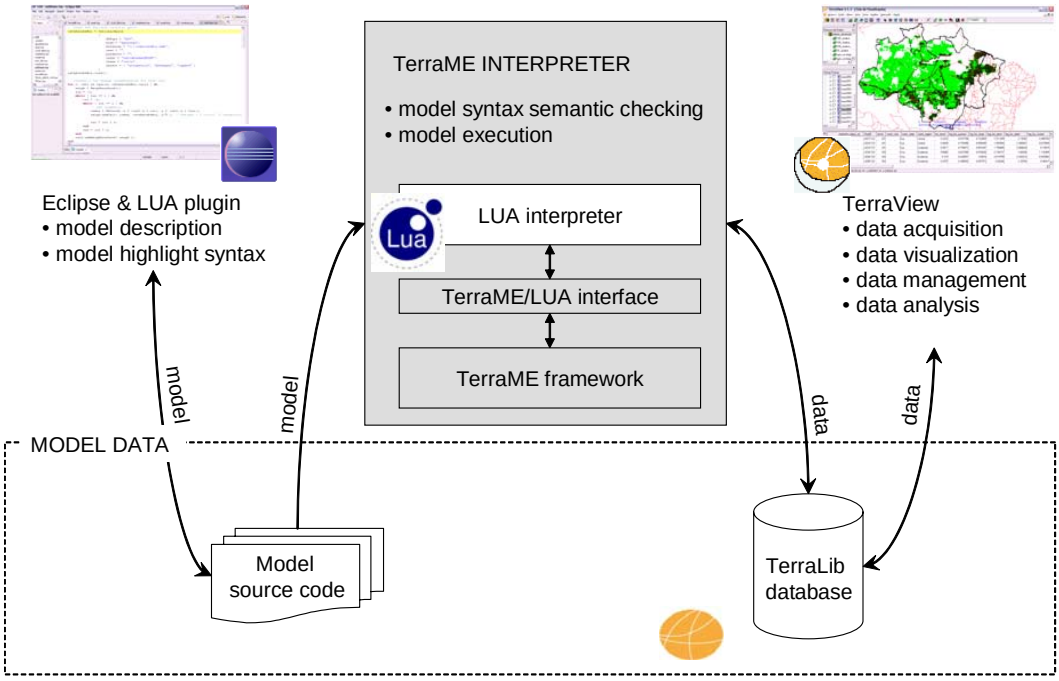


Figura 4.1 – TerraME modules and services.

The LUCC modeling services provided by *TerraME* architecture are distributed in software modules (Figure 4.1). Two of these modules were developed in this work: the *TerraME Framework* and the *TerraME* interpreter. The other modules are the *TerraLib* GIS library and the *TerraView* GIS application (Câmara, Souza et al. 2000), the LUA programming language (Ierusalimschy, Figueiredo et al. 1996), and Eclipse software development platform (Bott 1989).

The modeler can use any text editor to develop its models in *TerraME modeling language*. Preferably, she should use the Eclipse software development platform, which has a free *plugin* for LUA that can be configured to invoke the *TerraME interpreter*. This way, the modeler develops, executes and debugs the model inside an integrated environment.

The model source code is sent to the *TerraME interpreter*. The *TerraME interpreter* is the application that put all modeling services together, providing syntax and semantic checking, model simulation, and model assessment. It receives a set of text files containing models described in *TerraME Modeling Language* and executes them in the

order they have been passed as parameters. The *TerraME/LUA interface* registers new types for spatial dynamic modeling in the *LUA interpreter* virtual machine. The *LUA interpreter* calls functions provided by the *TerraME framework* which implements these types.

For data management and analysis, *TerraME* reads model input data from and saves model output to *TerraLib* spatiotemporal database. This database can be constructed using the *TerraView* application. The *TerraLib API* provides several methods for computation of cell attribute values from raster and vector data. *TerraView* provides spatial query and spatial statistical functionalities.

4.4 TerraME system architecture

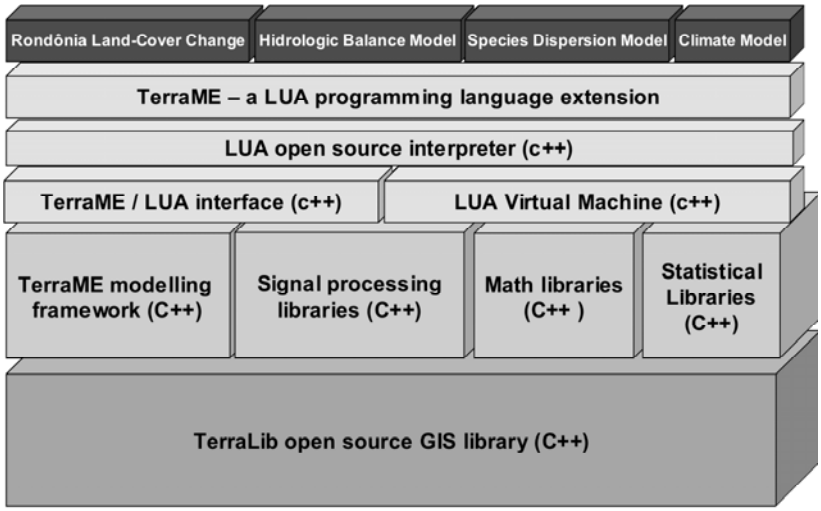


Figura 4.2 – TerraME modeling environment architecture.

Figure 4.2 shows the layered *TerraME* architecture. Lower layers provide basic services over which upper layer services are implemented. In the first layer, *TerraLib* offers typical GIS spatial data management and analysis services, and additional functions for temporal data handling. The *TerraME framework* is the dynamic modeling architecture core implemented in this work. It provides the simulation engine and the calibration and validation services. It is an open source ANSI C++ implementation of the nested-CA model, portable for Windows and Unix-like operating systems. This framework can be used directly for model development. Since the development of models in C++ can be a challenge for non-programmers, *TerraME* provides a high-level modeling language.

The third layer of the architecture implements the *TerraME modeling language* interpreter and runtime environment. The *TerraME*/*LUA* interface extends *LUA* with new data types for spatial dynamic modeling and services for model simulation and evaluation. Using the *LUA* library API, it exports the *TerraME framework* API to the *LUA* interpreter so that it recognizes the *TerraME* types. If required, other C or C++ applications (such as statistical libraries) can have their APIs exported to the LUA interpreter and integrated in the architecture. The last layer, called application layer, is where the end user models are located.

4.5 The TerraME framework architecture

We compared two alternative software architectures for the *TerraME* core. If developed as a library of spatial dynamic modeling classes and objects, the *TerraME* core would be very flexible and would not impose a rigid structure for the applications that reuse it. The drawback is that it would have a steep learning curve and it would be more difficult to develop new models. For better reuse, *TerraME* has been developed as a framework to capture the common design decisions in the development of spatial dynamic models. A framework is more expressive than a library, allowing more efficient prototyping (Gamma, Helm et al. 1994; Schmidt, Fayad et al. 1996; Buschmann, Meunier et al. 1996).

To reuse a library, a programmer writes the main application code from where library objects are instantiated and functions are invoked. Using a framework, she reuses the framework architecture that calls user defined functions. A *TerraME Framework* application is a discrete-event simulator (Zeigler, Kim et al. 2005) for the nested-CA model. Using the *TerraME framework* API, the modeler defines the variables and rules that represent the spatial, analytical and temporal aspects of all scales of the model. Then, the *TerraME Framework* simulation engine executes the model, providing services for scale and automata scheduling and synchronization.

These services provided by the *TerraME framework* includes: (a) data structures for storage of model representation in memory; (b) a virtual machine that executes the model representation, and (c) analysis methods for assessing the model results.

4.5.1 Model representation services

The *TerraME Framework* building block is the type *Scale*, which represents a spatial dynamic system. As shown in Figure 4.3, a *Scale* has been implemented a composite of *Models. Scales* can be nested, allowing multiscale model development. When executed, it simulates its internal *Scales* executing each model in chronological order. *Automaton* models (i. e. local and global automata) represent the biophysical and socio-economic systems or actors who cause the changes. When simulated, they evaluate their rules over the cellular space, possibly changing the cell attribute values. *CellularSpace* models define which properties will be accessible to the automata in each space location. *Timer* models determine the order in which the automata will be simulated. When executed, they advance the simulation clock and execute the models that must be simulated at that moment.

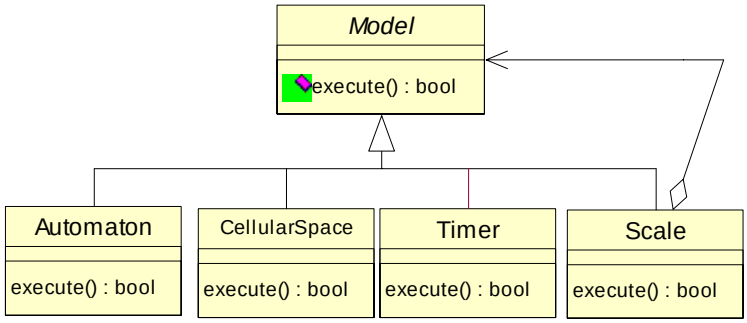


Figura 4.3 – UML diagram: *TerraME Framework* represents scale as a composite of models.

As shown in Figure 4.4, an *Automaton* has a set of *ControlModes.* Each *ControlMode* represents a discrete state of a hybrid automaton. It has two sets of rules: *JumpConditions* and *FlowConditions*. *JumpCondition* rules control a discrete state transition between *ControlModes*. *FlowCondition* rules describe the continuous behavior of an automaton in a *ControlMode*. For instantiating a rule, the modeler must inherit one of these classes and implement the abstract method *execute( ).*

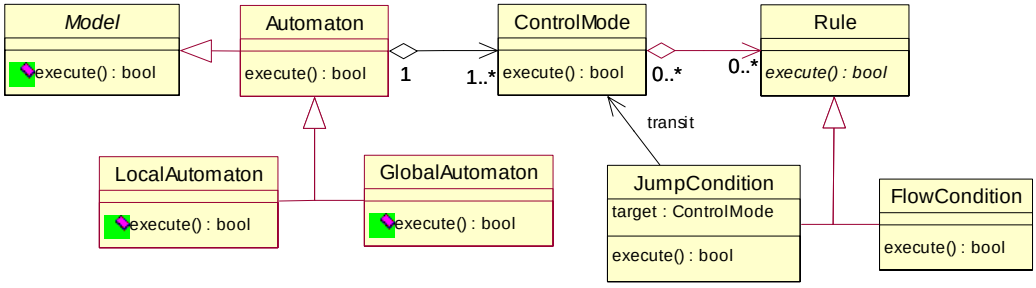


Figura 4.4 – UML diagram: *TerraME Framework* global and local automaton structure.

There are two kinds of automata. A *GlobalAutomaton* has the same active *ControlMode* for all *CellularSpace* locations. A *LocalAutomaton* has a different active *ControlMode* for each *Cell*. When an *Automaton* is simulated, it calls the method *execute*() of its active *ControlMode*. The active *ControlMode* executes its *JumpCondition* rules in the order they have been inserted. If a *JumpCondition execute()* method returns *true*, the *JumpCondition target* becomes the new active *ControlMode*. Then, the *JumpCondition* rules of the active *ControlMode* are evaluated. The process continues until it finds a *ControlMode* from which all *JumpCondition* rules return *false*. The *FlowConditions* of this *ControlMode* are executed in the order they have been inserted.

The *TerraME Framework* spatial model is formed by three components: (a) a cellular space providing, in certain spatial resolution, attributes describing the space (e.g., soils, climatic, socio-economic, etc.); (b) one or more alternative neighborhood relationships between the cells (e.g., Moore, Euclidian distance, or network connection, etc.); (c) one or more alternative spatial iterators for describing a trajectory that indicates the order that a cellular space shall be traversed by an *Automaton* when it is simulated. Some examples of trajectories are: northeast to southwest; concentric from a given *Cell* (urban centre), ascending order of the values of a given *Cell* attribute (deforestation potential).

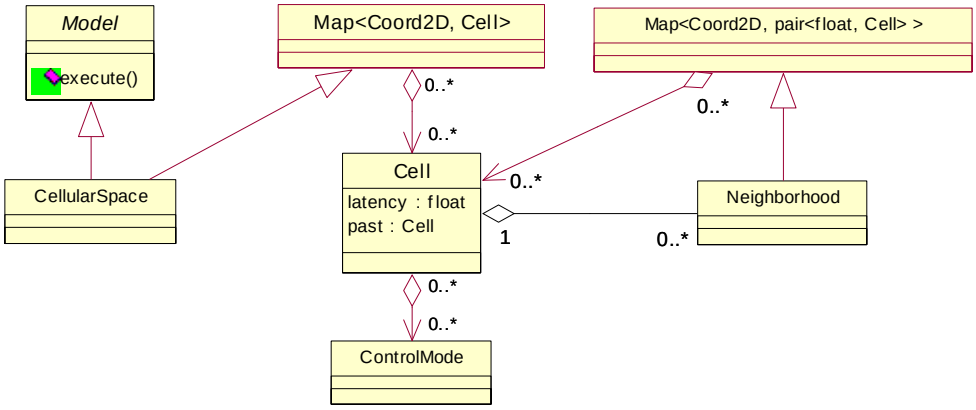


Figure 4.5 – UML diagram: *TerraME Framework* cellular space structure.

Figure 4.5 shows the UML diagram of the *CellularSpace* and *Cell* classes. A *CellularSpace* uses the C++ *Map*<T_1, T_2 > parameterized class, which implements a table for mapping objects of type T_1 in objects of type T_2. The *CellularSpace* is a *Map<Coord2D, Cell>* that maps 2D coordinates into *Cells*. Each *Cell* can have several alternative *Neighborhoods*. Each *Neighborhood* is a *Map<Coord2D, (float, Cell)>* that maps 2D coordinates in pairs (*weight*, *neigh*), where *weight* is the intensity of the relationship of the current cell to the cell *neigh.* Each *Cell* has two attributes: *past* – a copy of the cell attribute past values, and *latency* – the period of time since the last change in any cell attribute value. A *Cell* keeps track of all active *ControlMode* associated of all *LocalAutomaton* models. This way, it is possible to know the current discrete state of any *LocalAutomaton* in each *Cell*.

Figure 4.6 – UML diagram: *TerraME Framework* spatial iterator structure.

The modeler can define a *SpatialIterator* to represent a trajectory in a *CellularSpace* as an instance *Map<T, Cell>* that maps objects of type *T* in *Cell* objects. The modeler needs to provide the operator *<:T×T→{true, false}* for each iterator class. Figure 4.6 shows a *SpatialIterator* that maps *float* values (e. g. deforestation potential) in *Cell* objects.

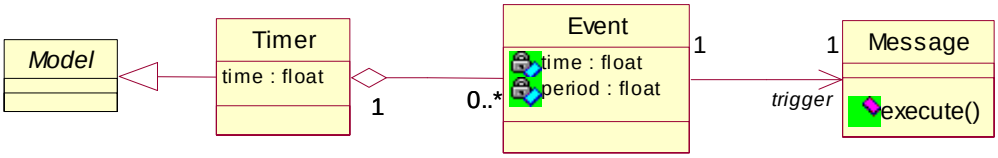


Figure 4.7 – UML diagram: *TerraME Framework* spatial iterator structure.

A *Timer* is a discrete-event scheduler. It has a set of chronologically ordered *Event* objects. As shown in Figure 4.7, each *Event* has two attributes: *time* – the simulation clock time that the *Event* will occur, and *period* – the periodicity in which the *Event* must *occur*. Each *Event* has an associated *Message*. A *Message* has an *execute()* method that must be implemented by the modeler. It will be used for calling functions from the *TerraME framework* API to request services during the simulation.

When a *Timer* is executed, it removes its first *Event*, updates the internal simulation clock (*Timer.time = Event.time)* and calls the method *execute()* from the *Message* associated to the *Event*. If this *execute( )* method returns *true*, the *time* attribute of the *Event* is updated (*time = time + period*) and the *Event* is reinserted in the *Timer*.

4.5.2 Model simulation services

The simulation services in *TerraME* are controlled by the *Timer* type, where each *Timer* implements a discrete-event scheduler. Each *Scale* can have several *Timers*. To keep the *Events* of all *Timer* objects ordered inside each *Scale*, the *TerraME* virtual machine maintains the *Timers* for each *Scale* in a balanced binary tree, called *Timer tree.* This tree is indexed by the time of the first *Event* of each *Timer*, as shown in Figure 4.8.a.

Since *Scales* can be nested, the *Events* of all nested *Scales* are chronologically ordered. The *Scales* are stored in a *Scale tree* that is indexed in chronological order of the first *Event* of its associated *Timer tree* (Figure 4.8.b). When the machine executes an *Event*, the associated *Timer* is removed from the *Timer tree* and the associated *Scale* is removed from *Scale* tree. The *Message* attached to *Event* is executed and the *Timer* and *Scale* objects are reinserted in the data structures. This process keeps *Scale*, *Timer*, and *Event* objects in a chronological order during simulation.

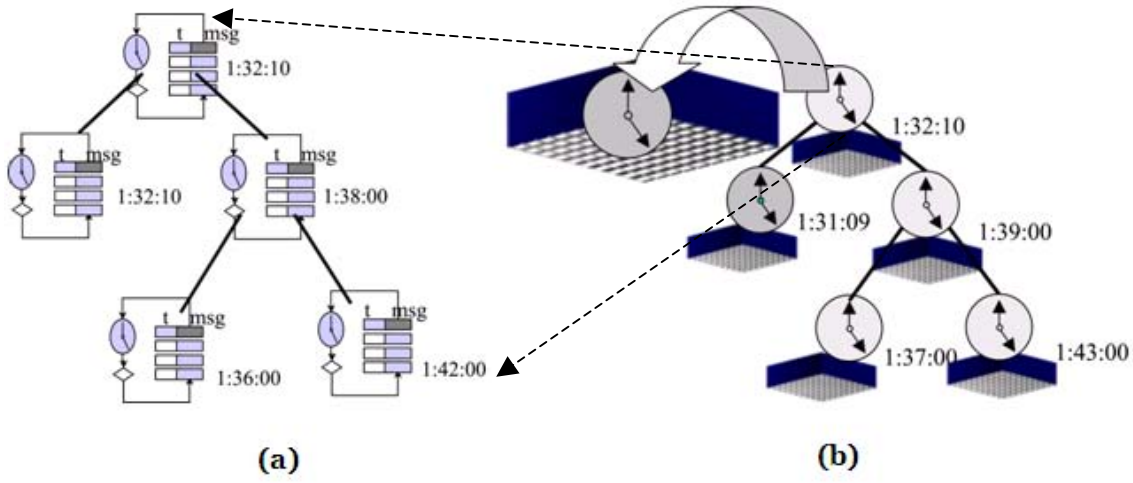


Figure 4.8 – *TerraME* scheduling data structures: *Timer tree* (a) and *Scale tree* (b).

4.6 The TerraME modeling language

The *TerraME Modeling Language* is a *LUA* extension. *LUA* is a dynamically typed language: variables do not have types; only values do. There are no type definitions in *TerraME*. The basic value types are *number* (double) and *string*. The value *nil* is different from any other value in the language and has the type *nil*. Functions in *LUA* are first class values. That is, a function definition creates a value of type *function* that can be stored in variables, passed as arguments to other *functions* and returned as results. The only structured data type LUA is *table*. It implements associative arrays, that is, arrays that can be indexed not only with integers, but with *string*, *double*, *table*, or *function* values. For *table* indexing, both *table.name* and *table["name"]* are acceptable. *Tables* can be used to implement records, arrays, and recursive data types. They also provide some object oriented facilities, such as methods with dynamic dispatching (Ierusalimschy, Figueiredo et al. 1996).

```
cell = { cover = "forest", distRoad = 0.3, distUrban = 2 };
cell.desfPot = cell.distRoad + cell[ "distUrban" ];
...
cell.reset = function( self )
               self.cover = "";  self.distRoad = 0.0; self.distUrban = 0.0;
               end
for i=1,10,1 do cellularSpace:add (cell); end
...
ForEachCell( cellularSpace, (function( cell ) cell:reset(); end ));
```

Figure 4.9 – The use of associative table and function values in *LUA*.

Figure 4.9 shows the use of *table* and *function* values. A *table* with three attributes (land cover, road distance, and urban centre distance) is created and stored in the variable *cell*. A new attribute is calculated and added to *cell* (deforestation potential is the sum of the road and urban center distances). A second attribute called *reset is* added to *cell*, defined as a *function* that receives the *table self* as parameter. Then, a cellular space of 10 cells is created, using the *TerraME* utility function *add*. Finally, the program calls the *TerraME* utility function *ForEachCell*, which traverses a *CellularSpace* and applies the *reset* function to each cell. The token ":" is a syntactic mechanism for method invocation: the modeler can write *table:name(…)* instead of *table.name(table, …)*. As a result, all cells are reset.

LUA has a powerful syntactical mechanism, called *constructor*, which provides an abstraction similar to the concept of *object* in the object oriented paradigm. When the modeler writes *name{…}*, the *LUA interpreter* replaces it by *name({… })*, passing the table *{…}* as a parameter to the function *name( )*. This function typically initializes, checks properties values and adds auxiliary data structure or methods (Ierusalimschy, Figueiredo et al. 1996). In figure 4.10, this mechanism is used to construct the "type" *MyCell*. When the table *c* is instantiated, the *distRoad* property value is corrected.

```
function MyCell( table )
        if( table.distRoad < 0 ) then table.distRoad = 0; end
        return table;
end
...
c = MyCell{..., distRoad = -0.1, ... }
```

Figure 4.10 – The use of the *constructor* mechanism in *LUA*.

To allow the description of spatial dynamic model as nested CAs, we included several new value types in *LUA* using the constructor mechanism. These values are: *Scale*, *CellularSpace*, *Cell*, *Neighborhood*, *SpatialIterator*, *GlobalAutomaton*, *LocalAutomaton*, *ControlMode*, *JumpCondition*, *FlowCondition*, *Timer*, *Event* and *Message*. We describe each type and its operations in what follows. The *TerraME* implementation of the hydrological balance model described in section 3.6.1 is used to exemplify the use of all values. For a single scale, this model simulates the rain water being drained according to the 9x9 km terrain digital model of a small village in Minas Gerais state, Brazil, called "Cabeça de Boi".

4.6.1 The multiple scale model

Multiple scales models can be developed by nesting several *Scales* values. A *Scale* represents a spatial dynamic system in a specific extent and resolution, for instance, the LUCC system. It models all analytical, spatial, and temporal aspects of the system.

4.6.1.1 The *Scale* type

A *Scale* is a container for automata, cellular spaces and timers, as shown in Figure 4.11. The automata represent the actors or processes that change the space. The cellular spaces represent the properties in each location. Timers define the order in which the automata are simulated. The modeler can add any finite number of *Scale*, *CellularSpace*, *LocalAutomaton*, *GlobalAutomaton* and *Timer* values to a *Scale*. All *Scale* values have an identifier to help in model debugging.

```
myScale = Scale{

          id = "MyScale",

          -- Add cellular spaces to this scale (spatial scale dimension)
          cs1 = CellularSpace{ … },
          cs2 = CellularSpace{ … },
          …
          csN = CellularSpace{ … },

          -- Add automata to this scale (analytical scale dimension)
          aut1 = LocalAutomaton{ … },
          aut2 = GlobalAutomaton{ … },
          …
          autN = LocalAutomaton{ … },

          -- Add timers to this scale (temporal scale dimension)
          t1 = Timer{ … },
          t2 = Timer{ … },
          …
          tN = Timer{ … },

          -- Add subscale to this scale (multiple scale modeling)
          sc1 = Scale{ … },
          sc2 = Scale{ … },
          …
          scN = Scale{ … },

}
```

Figure 4.11 – Defining Scales in TerraME Modeling Language.

In Figure 4.12, a *Scale* value is defined and stored in the variable *cabecaDeBoi*. A *CellularSpace* is added to the *Scale* to model the terrain. A *GlobalAutomaton* models the rain and a *LocalAutomaton* models the hydrologic balance process. A *Timer* defines when the automata is executed.

```
-- The "Cabeça de Boi" spatial dynamic model
cabecaDeBoi = Scale{

        id = "CabecaDeBoi",

        -- Add cellular spaces to this scale (spatial scale dimension)
        csCabecaDeBoi = CellularSpace{ … },

        -- Add global and local automata to this scale (analytical scale dimension)
        autRain = GlobalAutomaton{ … },
        autHidBalance = LocalAutomaton{ … },

        -- Add timers to this scale (temporal scale dimension)
        t = Timer{ … }

}
```

Figure 4.12 – A spatial dynamic hydrologic model in *TerraME Modeling Language*.

4.6.2 The spatial model

The *TerraME modeling language* spatial model provides three different types: *CellularSpace*, *Cell*, and *Neighborhood*.

4.6.2.1 The *CellularSpace* type

A *CellularSpace* is a multivalued set of *Cells* that is associated to a *TerraLib* spatiotemporal database. The modeler should specify the properties of the *CellularSpace* before using it. The *host* and *database* values indicate where the input data is stored. The *dbType* property identifies the database management system (*MySQL*, *PostgreSQL*, etc). The *layer* and *theme* properties are the names of the *TerraLib* database *layer* and *theme* that are used as input data. A *theme* is a *TerraLib* database structure that contains a set of objects. These objects are selected using a database query function over their attribute values, spatial relations, and temporal relations. The *select* property contains the names of the cell attributes loaded into the model from the input data set. The property *where* is used to filter the data, as in SQL statements. The *select* and *where* properties are optional.

```
-- Loads the TerraLib cellular space
csCabecaDeBoi = CellularSpace {
         dbType = "MySQL",
         host = "localhost",
         database = "CabecaDeBoi ",
         user = "",
         password = "",
         layer = "cells90x90",
         theme = "cells",
         select = { "altitude", "infCap" }
         where = "mask <> 'noData'";
}
```

Figure 4.13 – Defining a CellularSpace in TerraME Modeling Language.

In Figure 4.13, the "csCabecaDeBoi" *CellularSpace* is linked to the "cells" *theme* from the "cells90x90" *layer* of the "CabecaDeBoi" *TerraLib* database. For each cell, two attributes are loaded: elevation (*altitude*) and infiltration capacity (*infCap*). Only cells whose "mask" attribute value is different from "noData" will be loaded in the *CellularSpace*.

A *CellularSpace* has a special attribute called *cells*. It is a one-dimensional *table* of references for each *Cell* in the *CellularSpace*. The first *Cell* index is 1. Figure 4.14 shows how i-th *Cell* from a *CellularSpace* is referenced.

```
-- c is the seventh cell in the cellular space
c = csCabecaDeBoi.cells[ 7 ];

-- Five equivalent ways of update the attribute "infcap" from the seventh cell
c.infcap = 0;
c["infCap"] = 0;
csCabecaDeBoi["cells"][7]["infCap"] = 0
csCabecaDeBoi.cells[7]["infCap"] = 0
csCabecaDeBoi.cells[7].infCap = 0
```

Figure 4.14 – Referencing Cells from a CellularSpace in TerraME Modeling Language.

4.6.2.2 The *Cell* type

A *Cell* represents a space location, its properties, and its nearness relationships. A *Cell* is a *table* that includes persistent and runtime attributes. The persistent attributes are loaded from and saved to the database. The runtime attributes exist only in memory during the model execution. Section 4.6.5 describes the database management routines. Section 4.6.6 shows how runtime attributes can be defined for all *TerraME* values. A *Cell* value has two special attributes: *latency* and *past*. The *latency* attribute registers the period of time since the last change in a cell attribute value. It is used for rules that depend of how long the cell remains in a state. The *past* attribute is a copy of all cell

attribute values in the instant of the last change. For example, Figure 4.15 shows the command “if the cell cover is *abandoned land* during 10 year then the cover transit to *secondary forest”*. Figure 4.15 also shows a rule for simulating rain in a cell, which adds 2mm of water to the past amount of water in the cell.

```
if( cell.cover == "abandoned" and cell.latency >= 10 ) then cell.cover = "secFor"; end
…
cell.water = cell.past.water + 2;
```

Figure 4.15 – In TerraME cells have two especial attributes: *latency* and *past*.

4.6.2.3 The *Neighborhood* type

Each cell has one or more *Neighborhoods* to represent proximity relations. A *Neighborhood* is a set of pairs *(weight, cell)*, where *cell* is a neighbor *Cell* and *weight* is the neighborhood relationship strength. Figure 4.16 shows two equivalent pieces of code to traverse a cell neighborhood.

```
n = cell:getNeighborhood(1);
n:first();
while( not n:isLast() ) do
        neigh = n:getNeighbor();
        print( neigh.distRoad );
        print( n:getWeight() );
        n:next();
end
…
ForEachNeighbor(
        cell, 1,
        function( cell, neigh)
              print( neigh.distRoad );
              print( neigh:getWeight() );
        end
);
```

Figure 4.16 – Traversing a Neighborhood in TerraME Modeling Language.

The method *getNeighborhood(index)* of a *Cell* value recovers its i-th *Neighborhood*. A *Neighborhood* has several methods. The methods *first( )* and *last( )* point to the first and last neighborhood cell. The methods *next()* and *previous()* move back and forth. The methods *isFirst()* and *isLast()* return *true* if the current neighbor is the first or last neighbor, respectively. The method *getNeighbor()* returns the current neighbor *Cell*. The method *getWeight( )* returns the intensity of the neighborhood relationship between the cell and its current neighbor. *ForEachNeighbor* is a *TerraME* utility routine that receives a function as parameter and traverses the i-th *Neighborhood* of a *Cell* applying this function to all cells in it.

4.6.3 The analytical model

TerraME implements the nested-CA different models of computation for spatial process simulation, described in Section 3.4. The *GlobalAutomaton* model allows the development of models based on the *agent* approach. The *LocalAutomaton* model allows the development models based on a *cellular automata* approach. A *GlobalAutomaton* traverses a *CellularSpace* sequentially, evaluating its rules on each *Cell*. A *LocalAutomaton* has a copy of its internal state in each cell. Changes occur in parallel, all locations may change simultaneously.

4.6.3.1 The *GlobalAutomaton* and *LocalAutomaton* types

The *GlobalAutomaton* and *LocalAutomaton* types are containers of *ControlMode* and *SpatiaIterator* objects, as shown in Figures 4.17 and 4.18. A *ControlMode* represents a discrete state of the automaton. A *SpatialIterator* defines the spatial trajectory of the automaton. When an automaton is executed, it uses this trajectory to traverse a *CellularSpace* subset, visiting the *Cell* values in a predetermined order. At each *Cell*, the current *ControlMode* determines the set of possible actions (rules). The initial *ControlMode* of an automaton is the first one defined in its interior.

```
aut = GlobalAutomaton{

        SpatialIterator{…},
        SpatialIterator{…},
        …
        SpatialIterator{…},

          ControlMode{…},
          ControlMode{…},
          …
          ControlMode{…}
}
```

Figure 4.17 – Defining a GlobalAutomaton in TerraME Modeling Language.

```
aut = LocalAutomaton{

        SpatialIterator{…},
        SpatialIterator{…},
        …
        SpatialIterator{…},

          ControlMode{…},
          ControlMode{…},
          …
          ControlMode{…}
}
```

Figure 4.18 – Defining a LocalAutomaton in TerraME Modeling Language.

4.6.3.2 The *SpatialIterator* type

SpatialIterator values are useful to reproduce spatial patterns or represent process preferential directions (anisotropy). Even for *LocalAutomaton* values, which are parallel spatial processes, *SpatialIterators* are useful to define change suitability surfaces, which associate each *Cell* to a real number that indicates how prone the *Cell* is to specifics types of change (forest to pasture, pasture to abandonment, pasture to urban, etc).

A *SpatialIterator* is defined by three values. The first is the *CellularSpace* over which the trajectory will take place. The second value is a *function* that receives a *Cell* as parameter and returns a Boolean value. It is used to filter the *Cells*. If this *function* returns *true*, the cell is included in the trajectory. The third value is a *function* used to order this subset of *Cells*. It receives two *Cell* values as parameters and returns *true* if the first one is greater than the second. Figure 4.19 shows an example of *SpatialIterator* useful to simulate the deforestation process in LUCC models. The *SpatialIterator it* for the *CellularSpace cs* is defined by two *functions*. The first function select only cells whose land cover is “forest”. The second orders the *Cells* according to their distance to the nearest road, making *Cells* closer to roads more suitable to change. If the second *function* is not defined, the *Cells* are traversed from North to the South and from West to the East. If both *functions* are not defined, all *Cells* are included in the trajectory.

```
it = SpatialIterator{
        cs,
        function( cell ) return cell.cover == "forest"; end,
        function( c1, c2 ) return c1.distRoad > c2.distRoad; end
}
```

Figure 4.19 – Defining a *SpatialIterator* in TerraME Modeling Language.

4.6.3.3 The *ControlMode* type

A *ControlMode* is a container of two kinds of rules: *JumpCondition* and *FlowCondition*, Figure 4.20. A *JumpCondition* represents discrete state transition of an automaton. The *JumpConditions* of a *ControlMode* are executed in the order they have been defined. *FlowConditions* are rules that define behavior of the automaton in a specific state. The *FlowConditions* of a *ControlMode* are executed only if no *JumpCondition* has caused a

state transition. They are executed in the order they have been defined. All *ControlMode* has a unique identifier used by *JumpConditions* to adreess it.

```
ControlMode{
        id = "working",

        Jump{…},
        Jump{…},
        …
        Jump{…},

        Flow{…},
        Flow{…},
        …
        Flow{…}
}
```

Figure 4.20 – Defining a *ControlMode* in TerraME Modeling Language.

4.6.3.4 The *JumpCondition* type

```
Jump{
        function(event, automaton, cell)
              return cell.water>cell.capInf;
        end,
        target = "wet"
}
```

Figure 4.21 – Defining a *JumpCondition* in TerraME Modeling Language.

A *JumpCondition* is defined by two properties. The first property is a user defined function that must return a Boolean value. The second property, called *target*, is a string containing the identifier of the target *ControlMode*. If the user defined function returns *true*, the automaton goes to the *ControlMode* indicated by the *target* property. If the function returns *false*, the automaton stays in the current *ControlMode*. The *JumpCondition* function receives three parameters: the *event* that causes its execution, the automaton that owns the *JumpCondition*, and a *Cell* where the *JumpCondition* is being evaluated. Using these parameters, the user can define *JumpConditions* which depends on the current simulation time ("if (event.time > 1) then..."), automaton state ("if(automaton.age > 20) then..."), or spatial properties ("if (cell.distRoad > 10) then ..."). Figure 4.21 shows a *JumpCondition* that causes a transition to the *"wet" ControlMode* when the amount of water in the cell is grater than the cell infiltration capacity.

4.6.3.5 The *FlowCondtion* type

```
Flow{
        function(event, automaton, cell)
              cell.water = cell.past.water + 2;
        end
}
```

Figure 4.22 – Defining a *FlowCondition* in TerraME Modeling Language.

A *FlowCondition* is an user defined function that receives three parameters: the *event* that cause the *FlowCondition* execution, the automaton that owns the *FlowCondition*, and the *Cell* where the *FlowCondition* is being evaluated. Figure 4.22 shows a *FlowCondition* that add 2 units to the amount of water in a cell.

4.6.3.6 The hydrologic balance model example

To exemplify the use of the TerraME analytical models, Figures 4.23 and 4.24 show the definition of two automata used in the implementation of the hydrological balance model described in section 3.6.1. The *GlobalAutomaton* "agRain" simulates the rain phenomenon. It uses a spatial iterator that limits its actions to the cells whose elevation is greater or equal to 1500 meters. When executed, it adds 2 units to the past amount of water in each cell.

```
-- The rain GLOBAL automaton
agRain = GlobalAutomaton{

       it = SpatialIterator{
              csCabecaDeBoi,
              function( cell ) return (cell.altitude >= 1500); end
       },

         ControlMode{
              id = "working",
              Flow{
                     function(event, agent, cell)
                            cell.water = cell.past.water + 2;
                            return 0;
                     end
              }
         }
}
```

Figure 4.23 – Simulating the rain in *TerraME Modeling Language*.

```
-- The soil water balance LOCAL agent
agWaterBalance = LocalAutomaton{

        it = SpatialIterator{
                csCabecaDeBoi,
                function( cell ) return true; end
         },

         ControlMode{
                id = "dry",

                Jump{
                        function(event, automaton, cell)
                                return cell.water>cell.capInf;
                        end,
                        target = "wet"
                }
         },

         ControlMode{
                id = "wet",

                Jump{
                        function( event, automaton, cell )
                                return  cell.water<=cell.capInf;
                        end,
                        target = "dry"
                },

                Flow{
                        function( event, automaton, cell )

                                -- calculates the water overflow
                                overflow = cell.water - cell.capInf;
                                cell.water = cell.capInf;

                                -- how many neighbours are lower than the cell?
                                countNeigh = 0;
                                height = cell.altitude;
                                ForEachNeighbor(
                                        cell, 0,
                                        function( cell, neigh)
                                                if (cell~=neigh) and
                                                   (height>=neigh.altitude) then
                                                        countNeigh = countNeigh + 1;
                                                end
                                        end
                                );

                                -- send water to the neighbors
                                ForEachNeighbour(
                                        cell, 0,
                                        function( cell, neigh )
                                                if (cell~=neigh) and
                                                   ( height>=neigh.altitude) then
                                                        neigh.water =  neigh.water +
                                                                       overflow/countNeigh;
                                                 end;
                                        end
                                );
                        end
                }
         }
}
```

Figure 4.24 – Simulating the water balance process in *TerraME Modeling Language*.

The *LocalAutomaton* "agWaterBalance" simulates the water balance process. It has two *ControlModes*: *"dry"* and *"wet"*. In the "*dry*" *ControlMode* the automaton checks if the amount of water in a cell is greater than the cell infiltration capacity. If *true* the automaton transit to the "*wet" ControlMode*. Otherwise, it does nothing. In the "*wet*" *ControlMode* it first checks if it must transit to "*dry*" *ControlMode*. If the transition is not necessary, it calculates the surplus of water and equally divides the surplus to the lower neighbor cells.

4.6.4 The temporal model

The *TerraME* temporal model provides three types: *Timer*, *Event* and *Message*. A *Timer* maintains a queue of pairs (*Event*, *Message)* to control the simulation time. The pairs are ordered by the *Event* times. An *Event* represents a time instant when the simulation engine must execute some computation (*Message)*. A *Message* is a user defined function from where simulation engine services can be called. Among these services, there are services to load data from the database, to save data in the database, to execute a specific automaton, to synchronize a cellular space, and to check if an automaton has been well defined.

4.6.4.1 The *Timer* type

A *Timer* is a container for pairs *(Event*, *Message),* Figure 4.25. Any finite number of pairs (*Event,* Message) can be added to a *Timer*.

```
time = Timer{
        Pair{
              Event{ ... },
              Message{ ... }
        },
        Pair{
              Event{ ... },
              Message{ ... }
        },
        ...
        Pair{
              Event{ ... },
              Message{ ... }
        }
}
```

Figure 4.25 – Defining a Timer in TerraME Modeling Language.

4.6.4.2 The *Event* type

An *Event* is defined by two mandatory properties (*time* and *period*) and an optional one (*priority*). The *time* property defines the next instant of time (in the simulation clock) when the event must occur. The *period* property defines the periodicity in which the event must occur. The *priority* property is used to decide what event must occur first when two events have the same value for the *time* property. The default *priority* value is 0 (zero). Smaller values have higher priority. Figure 4.26 presents an *Event* that must occur at the year 1985, repeat every year, and has priority equal to -1.

```
Event{ time = 1985, period = 1, priority = -1 }
```

Figure 4.26 – Defining a Event in TerraME Modeling Language.

4.6.4.3 The *Message* type

A *Message* is an user defined function whose parameter is the *Event* that has caused its execution. Figure 4.27 shows a *Message* that prints the simulation time in the screen, executes the automaton "agRain", and prints the word "Rained" in the screen.

```
Message{
	function( event )
		print(event.time);
		agRain:execute( event );
		print("\tRained");
		return 0;
	end
}
```

Figure 4.27 – Defining a *Message* in TerraME Modeling Language.

4.6.5 Database management routines

A *TerraME CellularSpace* provides three functions for database management. The *load()* function loads the cell attributes from the spatial database. 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. Figure 4.28 shows how these functions are invoked for the *CellularSpace* called *csCabecaDeBoi*.

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

Figure 4.28 – Loading space attributes in *TerraME Modeling Language*.

The *TerraME CellularSpace* also provides a function to save the cell attribute values in the associated *TerraLib* database. Its syntax is *save (time, themeName, attrNameTable)*. The parameter *time* is the timestamp that will be associated to the data. The parameter *themeName* is the *TerraLib theme* where the data will be saved. The parameter *attrNameTable* is a table with the names of the cell attributes to be saved. If the third parameter is an empty table or a *nil* value, all cell attributes will be saved. When the *save(...)* function is executed, a *view* named *Result* is created in the *TerraLib* database and a *theme* containing the saved data is inserted in this *view*. The name of the theme is composed concatenating the parameters *themeName + time*. When the *save(...)* function is called with the parameters shown in Figure 4.29, the values of the attribute "water" of all cells from the *CellularSpace "csCabecaDeBoi"* are saved in the *themes*: "sim1985", "sim1986", "sim1987", and so on.

```
csCabecaDeBoi:save(event:getTime(),"sim", {"water"});
```

Figure 4.29 – Saving cell attributes values in *TerraME Modeling Language*.

4.6.6 Defining runtime variables

The user can define runtime variables for any *TerraME* type by defining it in a statement (*variable "."* *newVariable "="* *value*). One can define runtime variables for *Cell*, *LocalAutomaton*, or *Event* values. In Figure 4.30, the runtime variable "name" is added to an *Event* and receives the value "initialEvent", and a runtime attribute called "water" whose value is 0 (zero) is added to each cell from the CellularSpace "csCabecaDeBoi".

```
-- Creating new event attributes
ev = Event{ time = 1985, period = 1 };
ev.name = "initialEvent";

-- Creating new cell attributes
ForEachCell( csCabecaDeBoi, function( cell ) cell.water = 0; end );
```

Figure 4.30 – Defining a runtime attribute in *TerraME Modeling Language*.

4.6.7 Synchronizing the space

TerraME implements the nested-CA synchronization model described in section 3.5. The *Cell:synchronize(), CellularSpace::synchronize()* and *Scale:synchronize()* functions can be used for synchronization, as shown is Figure 4.31. The variable

"csCabecaDeBoi" is a 10 x 10 *CellularSpace* where the cover of each *Cell* is “forest”. The first time this code will be executed, all cells will be "deforested" and the sentence *"Number of deforested cells: 100"* will be printed. The second time, the sentence *"Number of deforested cells: 0"* will be printed because the cells have already been deforested. However, if the function *"cell:synchronize()"* had been excluded, the changes would not have been committed. Then, the output would always be *"Number of deforested cells: 100"*.

```
count = 0;
for i, cell ipairs( csCabecaDeBoi ) do
	if( cell.past.cover == "forest") then
		cell.cover = "deforested";
		count = count + 1;
	end
	cell:synchronize();
end
print("Number of deforested cells:"..count);
```

Figure 4.31 – Synchronizing a CellularSpace in TerraME Modeling Language.

4.6.8 Configuring and starting the simulation

Before starting the simulation, it is necessary to verify if the syntax of the model is correct. This requires that each automaton has all its *ControlModes* identified in the *target* properties of its *JumpConditions.* This verification can be performed through the function *build()* from both *LocalAutomaton* and *GlobalAutomaton* types. If there is a syntax error, the *build()* function aborts the model and prints an error message identifying the wrong *JumpCondition* target.

The simulation will start at the instant of the first *Event* value. It is necessary to configure the final simulation time. The function *Scale:config(finalTime)* serves this purpose. It receives the value of the final simulation time as its parameter. In Figure 4.32 the automata "*agRain*" and "*agWaterBalance*" from the *Scale* "*cabecaDeBoi*" are verified, the *Scale* is configured to stop at the year 1987, and the simulation is started when the function *executed()* from the *Scale* "cabecaDeBoi" is called.

```
cabecaDeBoi.agRain:build( );
cabecaDeBoi.agWaterBalance:build( );
cabecaDeBoi:config(1987);
cabecaDeBoi:execute();
```

Figure 4.32 – Configuring and starting the simulation in *TerraME* Modeling Language.

4.7 Comparison with previous work

This section compares the *TerraME* modeling environments with the most relevant platforms used to LUCC modeling: *Swarm*, *STELLA* and *GEONAMICA*. In opposition to *GEONAMICA* (Engelen, White et al. 1997) and *TerraME*, tools originally conceived to aid spatial dynamic modeling, *Swarm* (Minar, Burkhart et al. 1996) and *STELLA* (Roberts, Anderson et al. 1983), are based on non-spatial foundations: agent theory and system theory. *GEONAMICA* implements the *Layered-CA* model and *TerraME* uses the nested-CA model. All of these environments provide abstractions to allow problem decomposition. In *STELLA*, a system is a composition of other systems. In *Swarm*, objects *swarm* are containers for sets of agents that can be nested, forming a hierarchy of *swarm* objects. In *GEONAMICA*, *model building block* (MBB) objects can be composed of several MBBs. In TerraME, *Scales* can be nested for multiscale models.

The *STELLA* modeling tool (Roberts, Anderson et al. 1983) is an application that provides a graphical interface for model design: in the flow diagram, systems (rectangles) are connected by flows of energy (arrows). Systems are represented by a set of continuous variables, and input and output flows. The flows are represented by differential equations. The model is continuous, sequential, and predetermined by the modeler. It is not possible to represent processes whose behavior depends on external events. The spatial modeling framework *SME* (Maxwell and Costanza 1995) integrates a cellular space with GIS systems and embeds a *STELLA* model in each cell.

Swarm (Minar, Burkhart et al. 1996) is an open source library of classes and objects for the development of multiagent simulations. Actors and processes are modeled as communication agents. The modeler used the inheritance and dynamic binding mechanisms from the host programming language to extend the *Swarm* basic models. The model behavior is specified in the host programming language. The model needs to be recompiled each time its code is updated. Several discrete-event schedulers can be defined to coordinate agents in time, allowing multiple temporal extents and resolutions. The *Kenge* toolkit implements a GIS integrated cellular space for the *Swarm* platform (Box 2002).

GEONAMICA is a set of C++ templates and *ActiveX* components (Engelen, White et al. 1997) that depend on the object oriented properties (inheritance, dynamic binding) of the host language to be extended. A model building block (MBB) component represents an actor or a process. Models are described graphically in a system diagram where several MBB (rectangles) are connected by flows of information (arrows). For each MBB, the modeler should describe, in the host programming language, the rules that will be executed when four different types of events occur: *on read*, *init*, *step*, and *on write*. After this, *GEONAMICA* generates the source code of the described model. Then, the model is compiled and linked with the simulation engine. The *GEONAMICA* spatial model is integrated with a GIS.

The *STELLA*, *Swarm*, and *GEONAMICA* platforms do not satisfy the full requirements for multiscale LUCC modeling. They do not provide a special abstraction to represent the concept of *scale*. Their foundations are two restrictive to represent complex heterogeneous spatial dynamic models where different space partitions are represented in several scales. These platforms do not provide abstractions to reproduce the spatial patterns of change, or spatial process trajectories. Their analytical models do not distinguish between sequential and parallel spatial process, whereas *TerraME* provides the concept of *LocalAutomaton*, *GlobalAutomaton* and *SpatialIterator*. *GEONAMICA* and *Swarm* do not have special abstractions for continuous behavior modeling.

Swarm and *STELLA* do not provide methods for spatial model calibration and validation. The *GEONAMICA* framework provides methods that assess the model performance in several spatial resolutions. However, these methods do not distinguish between errors in the amount of change project by the model from errors on the location of the changes proposed by the model (Pontius 2002; Pontius, Huffaker et al. 2004).

4.8 Conclusion

In this work, we have presented the design and implementation issues involved in the development of a software platform for multiscale LUCC modeling. This software platform, called *TerraME*, implements the nested-CA model and services for spatiotemporal data analysis and management, model development, simulation, and

assessment. The *TerraME Modeling* language has been described in detail. Finally, the *TerraME* platform has been compared with relevant modeling tools used to LUCC model development: *Swarm*, *STELLA*, and *GEONAMICA*. The main contributions of *TerraME* are:

(a) The *Scale* model for representing, in a specific resolution and extent, all analytical, spatial, and temporal aspects of a geographical phenomenon. The nested *Scales* can be used to represent heterogeneous space, where each partition is characterized by different cell attributes, non-isotropic and non-stationary neighborhood relations, and processes or actors acting on specific temporal and spatial resolutions.

(b) The *LocalAutomaton* and *GlobalAutomatont* concepts, that enable the development of models that combine the agent-based and the cellular automata approaches. This allows the simulation of individual processes that change the space sequentially and of processes whose behavior is location dependent.

(c) The synchronization scheme allows the development of models where several sequential and parallel spatial processes or actors change the space in a asynchronous way.

(d) The *SpatialIterator* allows the representation of spatial trajectories and provides a mechanism to reproduce the spatial pattern of changes.

(e) The *TerraME* foundations allow the simulation of discrete, continuous, event-driven and situated behavior.

Since The *TerraME* is the only platform that satisfies all requirements of multiple scale modeling, we argue that *TerraME* is a suitable platform for *LUCC* modeling.

CHAPTER 5

APPLICATION OF NESTED CA FOR MODELING OF LAND USE CHANGE IN BRAZILIAN AMAZON

5.1 A brief review on LUCC modeling in Brazilian Amazon

This Chapter presents a review on LUCC modeling in Brazilian Amazon. To demonstrate the *nested*-CA properties, two multiple scale LUCC models have been implemented using the *TerraME* modeling environment. The main concepts of these models are briefly introduced. The general structures of their implementations in *TerraME* are presented. The simulation results are shown. In last sections, we highlight the mains contributions of the work and describe the future work directions.

One of the important areas of environmental change modeling is the Amazonia Rain Forest. The Brazilian Institute for Space Research (INPE) carries out a yearly comprehensive survey of deforested areas, using remote sensing images from the LANDSAT (30 m resolution) and CBERS (20 m resolution) satellites. From 1985 to 2005, INPE's data indicates that more than 350,000 km2 of forest have been converted to agriculture and pasture (INPE 2005). INPE's most recent results indicate a deforestation rate of 27.300 km^2 for the period August 2003- July 2004 and of 18.900 km^2 for the period August 2004- July 2005. In the extreme case, deforestation rates can be as high as 10.000 km^2 in a single month.

The process of change in Amazonia is relevant for global primary production[1] (Dixon, Brown et al. 1994; Malhi, Meir et al. 2002), for global biodiversity (Wilson 1989; Demiranda and Mattos 1992; Dale, Pearson et al. 1994). Deforestation in Amazonia has impacts on the public health system (Coura, Junqueira et al. 1994; Githeko, Lindsay et al. 2000; Vasconcelos, Travassos da Rosa et al. 2001), on atmospheric chemistry (Ganzeveld and Lelieveld 2004), on the climate system (Nobre, Sellers et al. 1991; Laurance and

[1] Primary production is the production of biological organic compounds from inorganic materials through photosynthesis or chemosynthesis. Organisms that can create biomass in this manner (notably plants) are known as primary producers, and form the basis of the food chain.

Williamson 2001; Werth and Avissar 2002; Oyama and Nobre 2003; Negri, Adler et al. 2004), and on global warming (Fearnside 1996).

LUCC studies have been made in the Amazon region in order to determine proximate causes and driving forces of deforestation (Pfaff 1999; Geist and Lambin 2002; Laurance, Albernaz et al. 2002; Aguiar, Kok et al. 2005; Fearnside 2005). LUCC models have been applied to the region in an attempt to understand the dynamics land use change dynamic and its consequences (Dale, Oneill et al. 1994; Pfaff 1999; Evans, Manire et al. 2001; Laurance, Cochrane et al. 2001; Soares, Assuncao et al. 2001; Soares, Cerqueira et al. 2002; Deadman, Robinson et al. 2004; Walker 2004; Walker, Drzyzga et al. 2004; Aguiar, Kok et al. 2005; Arima, Walker et al. 2005; Neeff, Graca et al. 2005).

There is currently no agreement as to the main causes of Amazon deforestation (Câmara, Aguiar et al. 2005). This is partly due to the lack of an established theory on human-environment interaction.

5.2 Applications

In this Section, we present the *TerraME* implementation of two multiple scale LUCC models:

- The *Conversion of Land Use and its Effects* (*CLUE*) (Veldkamp and Fresco 1996) model is applied to the Brazilian Amazon region (Figure 5.1), to simulate the deforestation process from 1997 to 2015. The allocation module is the *CLUE* model core. It answers the question *where* the demanded amount of change will take place (Verburg, Veldkamp et al. 1999). It includes two scales at which land use is allocated. A coarse scale is used to calculate the trends of the changes in land use pattern and to capture the influence of land use drivers that act over considerable distance. Based upon the pattern of land use change calculated at the coarse scale, but taking local constraints into account, the land use pattern is calculated at a finer scale. In this work, the allocation module had been

implemented in a generic way; so that it can be parameterized to be applied to other regions. The amount of change at each time step is a model parameter.

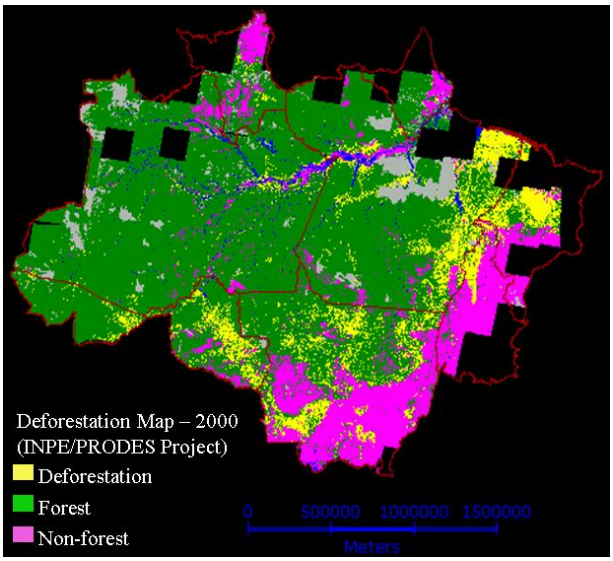


Figure 5.1 - Example 1: Legal Amazon study area, Brazil.
Source: INPE (2005)

- We have developed a LUCC model for the center-north region (Figure 5.2) of the Rondônia state, Brazil, which occupation history is associated to colonization projects created by the *Brazilian National Institute of Agrarian Reform* (INCRA), to induced migratory flows, to the BR-364 construction, and to the establishment of poles of development (Becker 1997). A TM/Landsat image series, from 1985 to 2000, agrarian maps, filed data, and census data have been used to partitioning the space in homogenous land units, called *occupation unit - UOP (Escada 2003)*. The homogenous space partitions have been delineated visually on the satellite images, defining regions formed by the repetition of texture elements, and linking different land cover patterns to different deforestation processes in specific temporal and spatial extents and resolutions. In this work, each *UOP* is represented as *Scale*. The whole study area is represented by a multiple scale model built by the composition of the *UOPs Scales*. In the right side of Figure 5.2, the *UOPs* are classified according to the farms size. Small farms *UOPs* appears in light blue. The large farms

UOPs are colored in dark blue. The medium farms UOPs are in an intermediary tonality. Urban center are in red, and reserve areas in green.

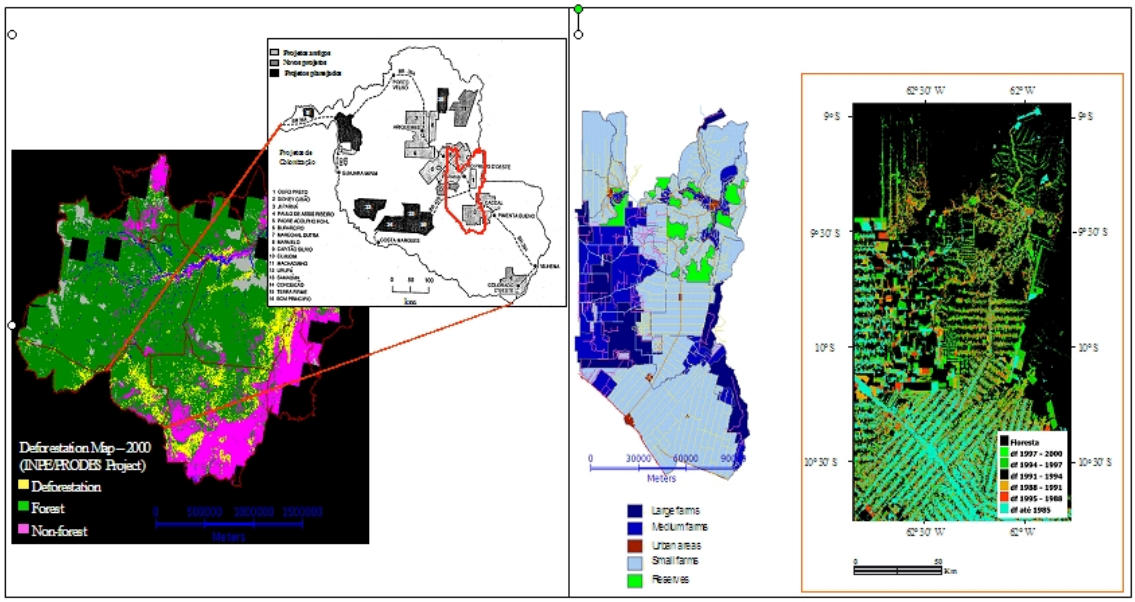


Figure 5.2 – Example 2: Rondônia study area, Brazil.
Source: adapted from Escada (2003).

5.2.1 The *CLUE* model in *TerraME*

In the CLUE model, the land cover is represented by the continuous variable $cover_{x,y,t,c}$ that records the proportion of each land cover type c in a cell (x, y) at the instant t. Aguiar (2005) has used a multiple regression method to analyze the descriptive data collected about the land use system at the two allocation scales, cell of 25×25 km^2 (local scale) and 100×100 km^2 (coarse scale), and at the instant t_0, initial time of the modeling exercise (1997). Figure 5.3 show the local and coarse allocation scales.

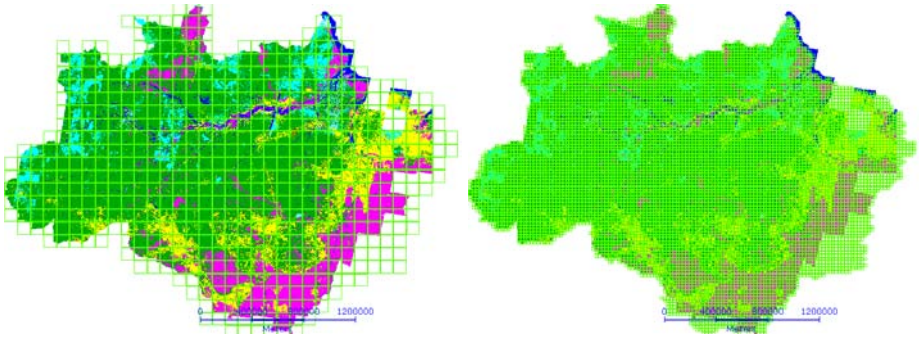


Figure 5.3 – Two allocation scales: cells of 100×100 km^2 (left), and cells of 25×25 km^2 (right).

As a result, for the year of 1997, two set of regressions equations have been obtained: one for each allocation scale. Each set of equations contains a regression equation that correlates the spatial pattern of each land use type *c* with other spatial attributes *i*: $cover_{x,y,t,c} = \beta_o + \beta_1 \cdot Attr_{x,y,t,1} + \beta_2 \cdot ATTR_{x,y,t,2} + \ldots$. This method has been used to identify the most important biophysical and socio-economic drivers of land use change (cell attributes which will be the dependent terms on the regression equations, $Attr_{x,y,t,i}$), as well as the quantitative relationships between these drivers and the surface area of the different land use types (the coefficient from the equations, β_i). At each simulation step, the first set of rules is used to allocate changes at the coarse scale. Then, the spatial pattern at the coarse scale is used with the second set of rules to allocate changes at the local scale.

```
allocationCLUE = Scale{

	id = "Amazon",

	landUseTypes = {
		"log_luc_pasture", "log_luc_temp", "log_luc_perm",
		"log_luc_nused", "log_luc_plant", "luc_forest"
	}

	landUseDrivers = {
		"conn_mkt", "log_dist_road", "prot_all", "agr_small",
		"log_setl", "log_dist_urban", "log_dist_mineral", "log_dist_river",
		"soils_fert_B1", "soils_fert_B3", "clima_humid", "log_dist_wood",
		"conn_port",
	}

	demand = {...},

	scLocal = Scale{

		regrParam = {...},

		cs = CellularSpace{...},
		aut = GlobalAutomaton{ ... },
		t = Timer{ ... }

	},

	scCoarse = Scale{

		regrParam = {...},

		cs = CellularSpace{...},
		aut = GlobalAutomaton{ ... },
		t = Timer{ ... }

	}

}
```

Figure 5.4 – *CLUE* allocation scales in *TerraME Modeling Language*.

Figure 5.4 shows the general structure of the CLUE allocation model represented in the *TerraME modeling language.* It is a scale composed by:

- A *landUseTypes* table that contains the name of each land use types in the input data. The land use categories in the input land use maps are: pasture, temporary agriculture, permanent agriculture, non-used land, and forest.

- A *landUseDrives* table that contains the name of each biophysical or socio-economic driver of land use change identified in the regression analysis. Aguiar (2005) have identified the following drivers: connection through roads to national markets, logarithm of the Euclidian distance to roads, percentage of

protected areas, percentage of small farms, logarithm of the number of settled families, logarithm of the Euclidian distance to urban centers, logarithm of the Euclidian distance to mineral deposits, logarithm of the Euclidian distance to large rivers, percentage of high and medium fertility soils area, average humidity in the three drier subsequent months of the year, logarithm of the Euclidean distance to wood extraction poles, and connection through roads network to main ports.

```
demand =  {
        { 26579657.13,  5218327.69, 1316347.75,  5331225.00, 234325.00, 252132617.44 },
        { 28406152.43,  5576919.64, 1406804.26,  5697575.00, 250427.30, 249474621.37 },
        { 30232647.74,  5935511.58, 1497260.77,  6063925.00, 266529.59, 246816625.31 },
        { 32059143.05,  6294103.53, 1587717.28,  6430275.01, 282631.89, 244158629.25 },
        { 33885638.36,  6652695.48, 1678173.79,  6796625.01, 298734.18, 241500633.18 },
        { 35712133.67,  7011287.43, 1768630.29,  7162975.01, 314836.48, 238842637.12 },
        { 37538628.97,  7369879.38, 1859086.80,  7529325.01, 330938.78, 236184641.05 },
        { 39365124.28,  7728471.33, 1949543.31,  7895675.02, 347041.07, 233526644.99 },
        { 41191619.59,  8087063.28, 2039999.82,  8262025.02, 363143.37, 230868648.93 },
        { 43018114.90,  8445655.22, 2130456.33,  8628375.02, 379245.67, 228210652.86 },
        { 44844610.21,  8804247.17, 2220912.84,  8994725.02, 395347.96, 225552656.80 },
        { 46671105.51,  9162839.12, 2311369.35,  9361075.03, 411450.26, 222894660.73 },
        { 48497600.82,  9521431.07, 2401825.86,  9727425.03, 427552.55, 220236664.67 },
        { 50324096.13,  9880023.02, 2492282.36, 10093775.03, 443654.85, 217578668.61 },
        { 52150591.44, 10238614.97, 2582738.87, 10460125.03, 459757.15, 214920672.54 },
        { 53977086.75, 10597206.91, 2673195.38, 10826475.03, 475859.44, 212262676.48 },
        { 55803582.06, 10955798.86, 2763651.89, 11192825.04, 491961.74, 209604680.41 },
        { 57630077.36, 11314390.81, 2854108.40, 11559175.04, 508064.04, 206946684.35 },
        { 59456572.67, 11672982.76, 2944564.91, 11925525.04, 524166.33, 204288688.29 }
}
```

Figure 5.5 – Model parameters: land use demand from each land use type from 1997 to 2015.

- A *demand* table that defines the total area demanded for each land use type at each simulation year, Figure 5.5. Each line of the *demand* table is associated to a specific year starting from 1997, and contains the total area (in m^2) required to each land use type in table *landUseTypes.*

- Two internal scales: *scLocal* and *scCoarse.* Each scale has a table to store the parameters of the regression equations, called *regrParam*, a *GlobalAutomata* that calculate the new spatial pattern for each land use type based on these parameters, a *CellularSpace* that store the percentage of each land use type and the values of each land use driver at each location, and a *Timer* that annually executes the *Automaton* and immediately synchronized the *CellularSpace.*

```
-- regrParam = {
--      land_use1 = { regrConstant, regError,
--                    { beta1, beta2, ..., betaN },
--                    { attr1, attr2, ..., betaN },
--                    log transformed? (true or false),
--                    elasticiy (defaul MIN_ELASTICITY),
--                    static? (0: dynamic, 1: static, -1: change towards demand dir.)
--      }
--      ...
-- }
```

Figure 5.6 – Format of the parameters of the regression equations for a scale.

Figure 5.6 presents the format of the table *regrParam*. For each land use type in *landUseType*, there is a line in the *regrParam* table that associates this land use type to a set (table) of parameters. The first parameter in this set is the value of the regression constant. The second is the value of regression error. A table containing the value of each regression coefficient is the third parameter. The fourth parameter is a table of indexes of the land use drives in the table *landUseDrivers*. To calculate the regression, the value of the land use driver at the *i-th* position in the fourth parameter table will be multiplied by coefficient in the same position in the third parameter table. The fifth parameter indicates whether the input data have been logarithm transformed during the regression analysis. The sixth parameter establishes the minimal elasticity accepted for the land use type (a *CLUE* parameter), and the seventh parameter indicates if the land use is static or dynamic (other CLUE parameter). Figures 5.7 shows the regression parameters used in this work for the *scLocal* allocation scale, a similar structure is used to the sc*Coarse* scale.

```
regrParamLocalScale = {

        log_luc_pasture = {
                3.958597, 0.63189,
                { -0.641055, -0.463439, -0.316293, 0.943192, -0.131481, 0.750784,
                0.328109, -0.052798 },
                { 1, 2, 3, 4, 6, 8, 9, 10},
                true,
                0.01,
                0
        },

        log_luc_temp = {
                1.498353, 0.49275,
                { -0.465827, -0.279017, 0.31186, 0.627869, 0.072989,-0.099603, 0.522392,
                0.500685, -0.03818, 3.384578 },
                { 1, 2, 3, 4, 6, 7, 8, 9, 10, 12 },
                true,
                0.01,
                0
        },

        log_luc_perm = {
                -1.45349, 0.36059,
                { -0.29376, -0.1618, 0.19072, 0.23094, -0.09049, 0.41573, 0.23233,
                4.44311 },
                { 1, 2, 3, 4, 7, 8, 9, 12 },
                true,
                0.01,
                -1
        },

        log_luc_nused = {
                1.981744, 0.46427,
                { -0.473846, -0.291643, 0.426325, 0.091095, -0.120919, 0.467058,
                0.379051, -0.03735, 5.962671 },
                { 1, 2, 4, 6, 7, 8, 9, 10, 12 },
                true,
                0.01,
                0
        },

        log_luc_plant = {
                -2.46878, 0.22932,
                { -0.12336, -0.06894, 0.04335, 0.16928, -0.0566, 0.0131, 0.14906,
                0.05723, 1.41639 },
                { 1, 2, 3, 4, 6, 7, 8, 9, 12 },
                true,
                0.01,
                -1
        },

        luc_forest = {
                -0.813838, 0.16291,
                { 0.148098, 0.071349, -0.091813, -0.187958, 0.020614, 0.026632,
                -0.239035, -0.062135, 0.014885 },
                { 1, 2, 3, 4, 6, 7, 8, 9, 10 },
                false,
                0.01,
                -1
        }
}
```

Figure 5.7 – Local scale regression parameters.

The figure 5.8 shows the model results for some time instants: 1999, 2005, 2010, and 2015. Changes are too concentrated the *Deforestation Arc*, and there is a little pressure on the *central* area.

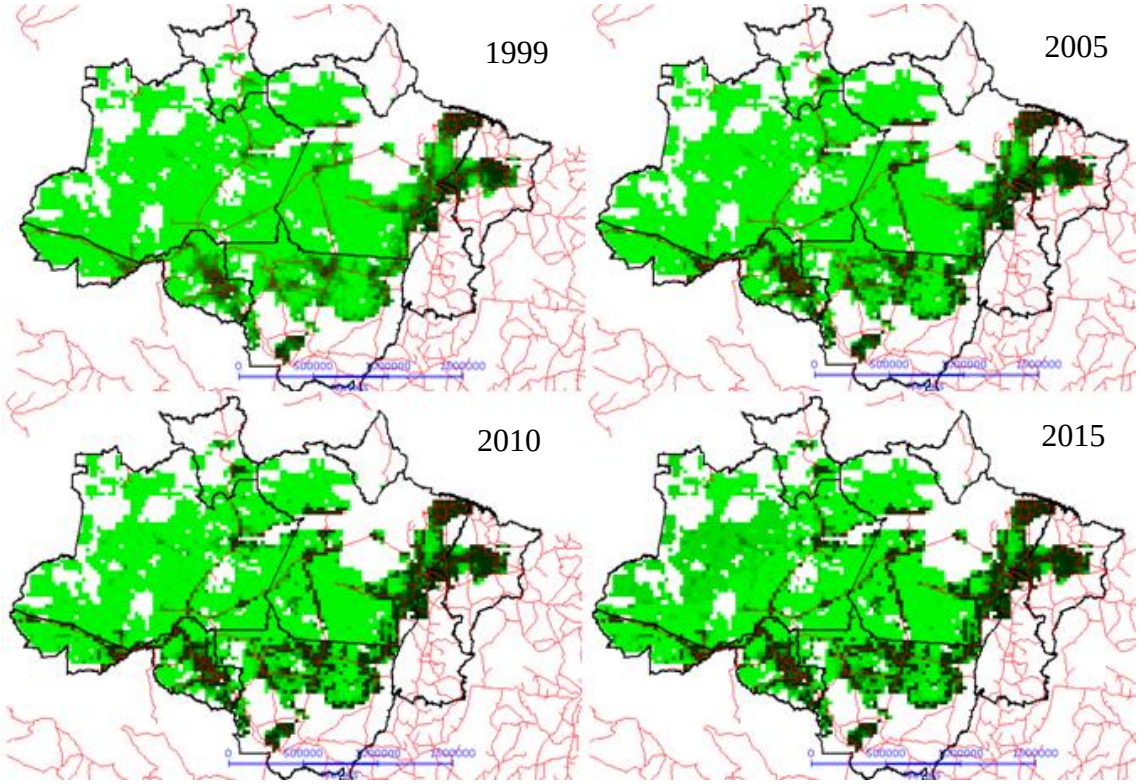


Figure 5.8 – CLUE results: deforestation process for the whole Brazilian Amazon region.

5.2.2 A deforestation model for heterogeneous spaces: the Rondônia case.

The main goal of developing this model is to test the *TerraME* modeling environment in the construction of multi-resolution models, with different actors, with distinct behaviour acting on contiguous space partitions. To accomplish this, we developed a deforestation model based on the assumption that small and large/medium farmers convert the forest to agriculture based on different behavioural rules, both for choosing the location of change and for defining the speed of change. In this work, we discuss only some of the main features of the model. A complete description of a full model being developed for this area using nested-CA, in the context of the GEOMA Project, is out of the scope of this work, and will be presented in future publication.

As farm properties and amount of capital available for different actors are so discrepant (small farms are less then 100 ha; medium from 100 to 1000 ha; large greater than 1000ha), we decided to test if different spatial resolutions would better represent the processes for different actors. The *TerraME Scale* type has been extended to generate two new types of *Scale* values: *smallScale* and *largeScale*. Then, each *UOP* has been represented as an instance of one of this *Scales* types, according to its classification: small farms area, or medium/large farms area.

The *smallScale* type is a composite of:

(a) A *Cellularspace* that has a categorical attribute to model the land cover, {forest, non-forest}, in each 500×500 m^2 cell.

(b) A *GlobalAutomaton* called *autSmallDemand* that calculates the rate of change based on the age of the INCRA settlement in the *UOP*, on the size of the land parcels, and on the installation of credit received from the Government in first years.

(c) Another *GlobalAutomaton* called *autSmallAllocation* that allocates the changes along the roads based on two spatial properties: the proximity to already established farmers through the roads network, and proximity to urban areas.

(d) A *Timer* is defined to every simulated year and executes the *autSmallDemand* automaton before the *autSmallAllocation* automaton execution.

One the other hand, the *largeScale* type is a composite of:

(a) A *Cellularspace* that has a continuous attribute to model the land cover, {percentage of forest}, in each 2500×2500 m^2 cell.

(b) A *GlobalAutomaton* called *autLargeDemand* that calculates the rate of change based on the age of the INCRA settlement in the *UOP* and on the size of the land parcels.

(c) Another *GlobalAutomaton* called *autLargeAllocation* that allocates the changes along the roads based on three spatial properties: the proximity to already established

farmers through the roads network, and the proximity to established farms which limits are in the same line of its frontiers (not necessarily where a road exists).

(d) A *Timer* is defined to every simulated year executes the *autLargeDemand* automaton before the *autLargeAllocation* automaton execution.

Figure 5.9 – Deforestation process in non-homogeneous space: forest (light gray) and deforest (dark gray).

Figure 5.9 illustrates two *UOPs*, one representing a small farms official settlement, established in 1985, called Vale do Anari (right); and another representing large farm area, being occupied since the 70ies, called Burareiro (left). Figure 5.10 presents the main differences between the automata *autSmallDemand* and *autLargeDemand*. Figure 5.11 describe the nearness relationships used by the automata *autSmallAllocation* (left) and *autLargeAllocation* (right). Figure 5.12 illustrates some simulation results. As the nested-CA model is a generic framework, several alternative space configurations and behavioural rules can be tested, allowing for a rich environment for hypothesis testing.

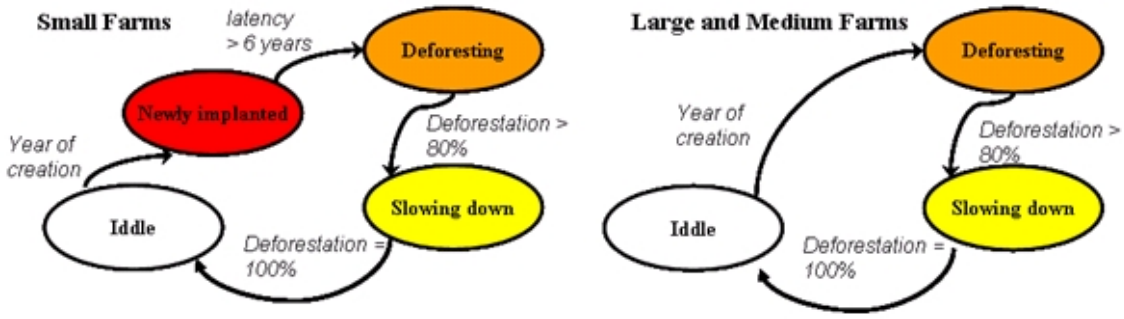


Figure 5.10 - Allocation module: The automata *autSmallDemand* (left) and *autLargeDemand* (right).

Figure 5.11 – Space partitions with alternative nearness relationships: roads (black lines), farms frontier line (light blue lines).

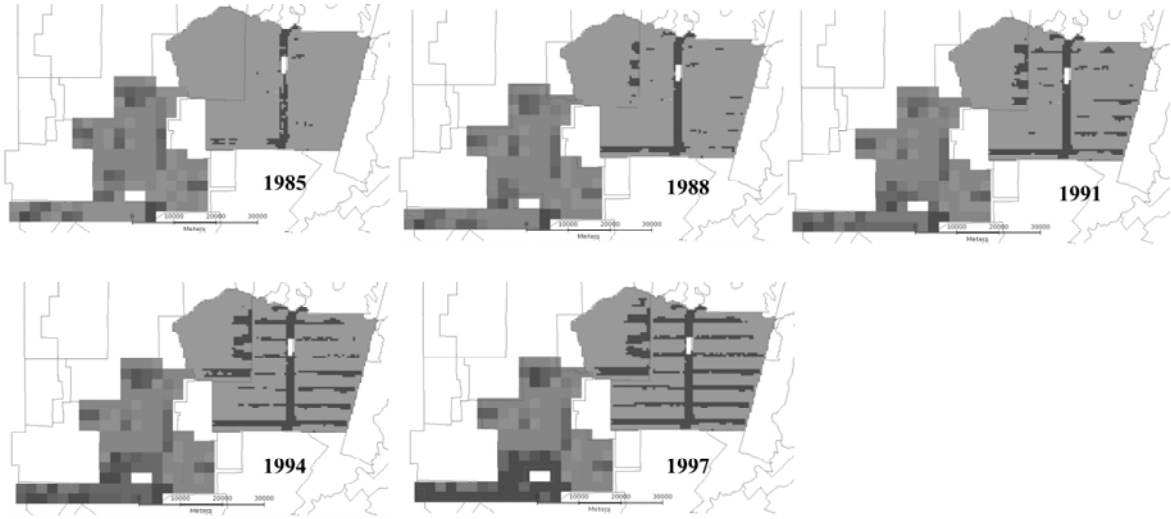


Figure 5.12 – Simulation results for deforestation process in Rondônia, Brasil, from 1985 to 1997.

In Figue 5.10, the *AutSmallDemand* automaton initial state is *Idle*. The automaton remains in this state until the simulation clock reaches the year of implantation of the *UOP*. Then, it transits to the *Newly Implanted* state, and establishes a higher

deforestation rate because the small farmers receive credit from the Government in the first 6 years after the settlement implantation. After this period, the automaton transits the Deforesting state, where the rate of deforestation is moderate and calculated according the UOP characteristics (age, parcel size). When the percentage of the total deforested are in the UOP is greater than 80%, the automaton transits to the *Slowing Down* state, where the deforestation rate progressively decrease, proportionally to the remaining amount of forest in the *UOP*. When there is no forest in the *UOP*, the automata transit to the state: *Idle*. rate of change is null while the UOP is not implanted. The *autLargeDemand* automaton does not have the *Newly Implanted* state because medium and large farmer do not receive credit.

5.3 Conclusion and future work

A model of computation, called *Nested Celluar Automata* (nested-CA), has been developed to support multiscale LUCC modeling. This model has been implemented in a modeling platform, called *TerraME*, which provides services for all stages of the spatial dynamic modeling process. Two multiscale LUCC models have been developed to test the nested-CA and the TerraME properties.

The nested-CA architecture facilitates integrated model developments, allowing complex dynamic spatial models to be constructed from hierarchically organized simple ones, in a *black box* fashion. It is possible to construct models in which different geographical space partitions are: inhabited by several specific actors and processes acting upon them in different spatial and temporal extents and resolutions; and are characterized by distinct local constraints and nearness relationship. It is possible to simulate discrete or continuous behavior, moving and communicating actors, and situated behavior. Neighborhood relations may be defined in alternative ways, including not only the conventional local relations, such as adjacency, Euclidean distance, etc., but also influence relations, such as connection through networks (e.g., roads or telecommunication), allowing for non-isotropic and non-stationary space relations.

The future work will be conducted within three different research areas:

- Models of computation for multiple scale spatial dynamic modeling: We believe that the nested-CA theoretical foundation needs to be explored to a better understanding of the nested-CA properties and their improvement. The situated behavior of the nested-CA models of computation has not been sufficiently investigated. It is necessary to perform experiments to test how this property can be used to represent the knowledge-based decision taking process. A map algebra over the cellular space can be developed to provide easier automata rules implementation.

- Software platforms for multiple scales LUCC modeling: The *TerraME* modeling environment will be in constant development. The short time projects are to parallelize the *TerraME framework* source code to obtain high performance computing, and to develop a visual interface where the modeler could describe the models graphically.

- Multiple scale LUCC model development: There is a huge demand for LUCC models for assess the land use system and for support the decision taking process. Using the *TerraME modeling environment*, we will continue to develop LUCC model to allow better understanding of the Brazilian Amazonian space and to support the planning of Government actions on this region.

REFERÊNCIAS BIBLIOGRÁFICAS

Aguiar, A.; G. Câmara, et al. Modeling Spatial Relations by Generalized Proximity Matrices. In: Brazilian Symposium in Geoinformatics, 5., 2003, Campos do Jordão, SP, Brazil. **Proceedings…**São José dos Campos: INPE, 2003.

Aguiar, A. P. D.; K. Kok, et al. Exploration of patterns of land-use change in the Brazilian Amazon using the CLUE framework. In: Open Meeting of the Human Dimensions of Global Environmental Change Research Community, 6., 2005, Bonn, Germany. **Proceedings…**Bonn: [s.n], 2005.

Almeida, C. **Spatial dynamic modelling as a planning tool: simulation of land use change in Bauru and Piracicaba (SP), Brazil**. 2003. 351p.(INPE-10567-TDI/942/A). Thesis (Doctorate in Remote Sense) - National Institute of Space Research (INPE), São Jose dos Campos, 2003.

Almeida, C. M.; A. M. V. Monteiro, et al. Empiricism and Stochastics in Cellular Automaton Modeling of Urban Land Use Dynamics. **Computers, Environment and Urban Systems**, v. 27, n. 5, p. 481-509, 2003.

Arima, E. Y., R. T. Walker, et al. Loggers and forest fragmentation: Behavioral models of road building in the Amazon basin. **Annals of the Association of American Geographers**, v. 95, n. 3, p. 525-541, 2005.

Batty, M. Modeling urban dynamics through GIS-based cellular automata. **Computers, Environment and Urban Systems**, v. 23, p.205-233, 1999.

Batty, M. Agents, cells, and cities: new representational models for simulating multiscale urban dynamics.**Environment and Planning A**, v. 37, n. 8, p. 1373-1394, 2005.

Becker, B. K. **Amazônia**. São Paulo, Brazil: Ática, 1997.

Binford, M.; Cassidy, L. Complementarity of Categorical and Continuous Approaches for Studying National level Development of Land-Use/Land-Cover Change in Thailand and Cambodia. In: Open Meeting of the Human Dimensions of Global Environmental Change Research Community, 6., 2005, Bonn, Germany. **Proceedings….** Bonn: [s.n], 2005.

Bott, F. **ECLIPSE an integrated project support environment**. Iee Computing Series, v. 14**.** London UK: IEEE Computer Society Press, 1989. 245p. ISBN:0-86341-169X.

Box, P. W. Spatial units as agents: Making the landscape an equal player in agent-based simulations. In: Gimblett, H. R. (ed). **Integration of agent-based modelling and geographic information systems.** London UK: Oxford University Press, 2002.

Briassoulis, H. **Analysis of land use change: theoretical and modeling approaches.** Lesvos, Greece: West Virginia University, 2000. Department of Geopgraphy - Regional Research Institute.

Buschmann, F.; R. Meunier, et al. **Pattern-oriented software architecture:** a system of patterns**.** John Wiley & Sons, 1996. ISBN: 978-0-471-95869-7.

Câmara, G.; A. P. Aguiar, et al. Amazonian deforestation models. **Science** v. 5712, n. 30, p. 1043-1044, 2005.

Câmara, G.; R. Souza, et al. TerraLib: Technology in Support of GIS Innovation. In: Brazilian Symposium in Geoinformatics, 2., 2000, São Paulo. **Proceedings...** São Paulo: [s.n], 2000.

Câmara, G.; L. Vinhas, et al. Design Patterns in GIS Development: The Terralib Experience. In: Workshop Brasileiro de Geoinformática - SBC, 3., 2001, Rio de Janeiro. **Proceedings...** Rio de Janeiro: SBC, 2001.

Costanza, R. Model Goodness of Fit - a Multiple Resolution Procedure. **Ecological Modelling**, v. 47, n. 3-4, p.199-215, 1989.

Couclelis, H. Cellular Worlds - a Framework for Modeling Micro-Macro Dynamics. **Environment and Planning A**, v. 17, n. 5, p. 585-596, 1985.

Couclelis, H. From cellular automata to urban models: New principles for model development and implementation. **Environment and Planning B-Planning & Design**, v. 24, n. 2, p. 165-174, 1997.

Couclelis, H. Chapter 2: Modeling frameworks, paradigms, and approaches. In: Clarke, K. C.; Parks, B. E.; Crane, M. P. (eds). **Geographic information systems and environmental modelling.** New York: Longman & Co, 2000.

Coura, J. R.; Junqueira, A. C., et al.. Chagas' disease in the Brazilian Amazon -A short review. **Revista do Instituto de Medicina Tropical de Sao Paulo**, v.36, n. 4, p. 363-368, 1994.

Dale, V. H.; Oniell, R. V., et al.. Modeling Effects of Land Management in the Brazilian Amazonian Settlement of Rondonia. **Conservation Biology**, v. 8, n. 1, p. 196-206, 1994.

Dale, V. H.; Pearson, S. M., et al.. Relating Patterns of Land-Use Change to Faunal Biodiversity in the Central Amazon. **Conservation Biology**, v. 8, n. 4, p. 1027-1036, 1994.

Deadman, P.; Robinson, D., et al. Colonist household decisionmaking and land-use change in the Amazon Rainforest: an agent-based simulation. **Environment and Planning B-Planning & Design**, v. 31, n. 5, p. 693-709, 2004.

Demiranda, E. E.; Mattos, C. Brazilian Rain-Forest Colonization and Biodiversity. **Agriculture Ecosystems & Environment**, v. 40, n. 1-4, p. 275-296, 1992.

Dixon, R. K.; Brown, S., et al.. Carbon Pools and Flux of Global Forest Ecosystems. **Science**, v. 263, n. 5144, p.185-190, 1994.

Engelen, G.; White, R., et al. Integrating Constrained Cellular Automata Models, GIS and Decision Support Tools for Urban Planning and Policy Making. In: Timmermans, H. (ed). **Decision support system in urban planning.** London, UK: E & FN Spon, p. 125-155, 1997.

Escada, M.. I. S. **Evolução de padrões de uso e cobertura da terra na região Centro-Norte de Rondônia**. 2003-04-14. 264 p. (INPE-10209-TDI/899). Tese (Doutorado em Sensoriamento Remoto) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos. 2003. Disponível em: <http://mtc-m12.sid.inpe.br/rep-/sid.inpe.br/jeferson/2003/06.30.13.31>. Acesso em: 15 maio 2007. rep: sid.inpe.br/jeferson/2003/06.30.13.31.

Escada, M. I. S.; Monteiro, A. M. V., et al. Análise de padrões e processos de ocupação para a contrução de modelos na Amazônia: Experimentos em Rondônia. In: Simpósio Brasileiro de Sensoriamento Remoto, 12., 2005, Goiânia, Brazil. **Proceedings…** São José dos Campos: INPE, 2005.

Evans, T. P.; Manire, A., et al. A dynamic model of household decision-making and parcel level landcover change in the eastern Amazon. **Ecological Modelling**, v. 143, n.1-2, p. 95-113, 1002.

Fearnside, P. M. Amazonian deforestation and global warming: Carbon stocks in vegetation replacing Brazil's Amazon forest. **Forest Ecology and Management** , v. 80, n. 1-3, p. 21-34, 1996.

Fearnside, P. M. Deforestation in Brazilian Amazonia: History, rates, and consequences. **Conservation Biology**, v. 19, n. 3, p. 680-688, 2005.

Forrester, J. W.. **Principles of systems**. Cambridge: MIT Press, 1968.

Gamma, E.; Helm, R. et al. **Design patterns:** elements of reusable object-oriented software. Reading, MA: Addison Wesley, 1994.

Ganzeveld, L.; Lelieveld, 2004. Impact of Amazonian deforestation on atmospheric chemistry. **Geophysical Research Letters**, v. 31, n.6, Art. n. L06105, 2004.

Geist, H. J.; Lambin. E. F.. Proximate causes and underlying driving forces of tropical deforestation. **Bioscience** v. 52, n. 2, p. 143-150, 2002.

Gibson, C. C.; Ostrom, E. et al.. The concept of scale and the human dimensions of global change: a survey. **Ecological Economics**, v. 32, n. 2, p. 217-239, 2000.

Githeko, A. K.; Lindsay, S. W. et al. Climate change and vector-borne diseases: a regional analysis. **Bulletin of the World Health Organization** .v. 78, n. 9, p. 1136-1147, 2000.

Henzinger, T. A. The Theory of Hybrid Automata. In: Symposium on Logic in Computer Science (LICS'96), 11., 1996, Washington. **Proceedings…** Washington: IEEE Computer Society, 1996.

Hestenes, D. Toward a Modeling Theory of Physics Instruction. **American Journal of Physics** v. 55, n. 5, p. 440-454, 1987.

Hopcroft, J. E.; Ullman, J. D. **Introduction to automata theory, language and computation**. Reading: Addison-Wesley Publishing Company, 1979.

Ierusalimschy, R.; Figueiredo, L. H. et al. Lua-an extensible extension language. **Software: Practice & Experience**, v. 26, p. 635-652, 1996.

Instituto Nacioanl de Pesquisas Espaciais (INPE). **Monitoramento da Floresta Amazônica Brasileira por Satélite: banco de dados PRODES**.2005. Avaible at: <http://www.obt.inpe.br/prodesdigital/>. Accessed in: 30 of March 2006.

Kok, K.; Veldkamp, A. Evaluating impact of spatial scales on land use pattern analysis in Central America. **Agriculture Ecosystems & Environment**, v. 85, n.1-3, p. 205-221, 2001.

Lambin, E. F.; Geist, H. J. et al. Dynamics of land-use and land-cover change in tropical regions. **Annual Review of Environment and Resources,** v. 28, p. 205-241, 2003.

Lambin, E. F.; Turner, B. L. et al. The causes of land-use and land-cover change: moving beyond the myths. **Global Environmental Change-Human and Policy Dimensions**, v. 11, n. 4, p. 261-269, 2001.

Laurance, W. F.; Albernaz, A. K. M. et al. Predictors of deforestation in the Brazilian Amazon. **Journal of Biogeography**, v. 29, n. 5-6, p. 737-748, 2002.

Laurance, W. F.; Cochrane, M. A. et al. The future of Brazilian Amazon. **Science**, v. 291, p.988, 2001.

Laurance, W. F.; Williamson, G. B. Positive feedbacks among forest fragmentation, drought, and climate change in the Amazon. **Conservation Biology**, v. 15, n. 6, p. 1529-1535, 2001.

Lim, K.; Deadman, P. J. et al.. Agent-based simulations of household decision making and land use change near Altamira, Brazil. In: Gimblett, R. (ed). **Integrating geographic information systems and agent-based modeling:** techniques for simulating social and ecological processes**.** New York: Oxford University Press, p. 277–310, 2002.

Malhi, Y.; Meir, P. et al. Forests, carbon and global climate. **Philosophical Transactions of the Royal Society of London Series A-Mathematical Physical and Engineering Sciences**, v. 360, p. 1567-1591, 2002.

Maxwell, T.; Costanza, R. Distributed modular spatial ecosystem modeling. **International Journal of Computer Simulation: Special Issue on Advanced Simulation Methodologies**, v. 5, n. 3, p. 247-262, 1995.

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

Minsky, L. M. **Computation:** finite and infinite machines. Englewood Cliffs, NJ: Prentice-Hall, Inc, 1967.

Munroe, D.; Calder, C. A. Multivariate multiple regression models of land use change: can land use drivers predict continuous land cover outcomes?. Open Meeting of the Human Dimensions of Global Environmental Change Research Community, 6., 2006, Bonn, Germany. **Proceedings…**Bonn: [s.n], 2006.

Neeff, T.; Graca, P. M. D. et al. Carbon budget estimation in Central Amazonia: Successional forest modeling from remote sensing data. **Remote Sensing of Environment**, v. 94, n. 4, p. 508-522, 2005.

Negri, A. J.; Adler, R. F. et al. The impact of Amazonian deforestation on dry season rainfall. **Journal of Climate**, v. 17, n. 6, p. 1306-1319, 2004.

Nobre, C. A.; Sellers, P. J. et al. Amazonian Deforestation and Regional Climate Change. **Journal of Climate**, v. 4, n. 10, p. 957-988, 1991.

O'Sullivan, D. Graph-cellular automata: a generalised discrete urban and regional model. **Environment and Planning B-Planning & Design**, v. 28, n. 5, p. 687-705, 2001.

Odum, H. T. **Systems ecology:** an introduction. John Wiley and Sons, 1983.

Oyama, M. D.; Nobre, C. A. A new climate-vegetation equilibrium state for Tropical South America. **Geophysical Research Letters**, v. 30, n. 23, 2003.

Parker, D. C.; Berger, T. et al. **Agent-based models of land-use and land-cover change:** report and review of an international workshop. Indiana:Indiana University. 2001. (L. R. No.6).

Pedrosa, B.; Câmara, G. et al. TerraML: a language to support spatial dynamic modeling. In: Congresso Brasileiro de GeoInformação, 2002. Caxambu. **Proceedings...** São José dos Campos: INPE, 2002.

Pfaff, A. S. P. What drives deforestation in the Brazilian Amazon? Evidence from satellite and socioeconomic data. **Journal of Environmental Economics and Management**, v. 37, n. 1, p. 26-43, 1999.

Pontius, R. G. Quantification error versus location error in comparison of categorical maps. **Photogrammetric Engineering and Remote Sensing**, v. 66, n. 8, p. 1011-1016, 2000.

Pontius, R. G. Statistical methods to partition effects of quantity and location during comparison of categorical maps at multiple resolutions. **Photogrammetric Engineering and Remote Sensing**, v. 68, n. 10, p. 1041-1049, 2002.

Pontius, R. G.; Huffaker, D. et al. Useful techniques of validation for spatially explicit land-change models. **Ecological Modelling**, v. 179, n. 4, p. 445-461, 2004.

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

Rosenschein, S. J.; Kaelbling, L. P.. A situated view of representation and control. **Artificial Intelligence**, v. 73, n. 149-73, 1995.

Russel, S. J.; Norvig, P. **Artificial intelligence –** a modern approach. Nova Jersey: Prentice Hall, 1995.

Schmidt, D. C.; Fayad, M. et al. Software pattern.. **Communications of the ACM**, v. 39, n. 10, p. 37-39, 1996.

Smyth, C. S. **A representational framework for geographic modeling.** Egenhofer, M. J.; Golledge, R. G. (eds.). New York: Oxford University Press, p. 191-213,1998.**(**Spatial and temporal reasoning in geographic information systems).

Soares, B. S.; Assunção, R. M. et al. Modeling the spatial transition probabilities of landscape dynamics in an amazonian colonization frontier. **Bioscience**, v. 51, n. 12, p. 1059-1067, 2001.

Soares, B. S.; Cerqueira, G. C.et al. DINAMICA - a stochastic cellular automata model designed to simulate the landscape dynamics in an Amazonian colonization frontier. **Ecological Modelling**, v. 154, n. 3, p. 217-235, 2002.

Southworth, J.; Binford, M. Footsteps on the land: detecting and analyzing change within a continuous landscape framework in and around National Park, Uganda. In: Open Meeting of the Human Dimensions of Global Environmental Change Research Community, Bonn, Germany, 2005. **Proceedings…** Bonn: IGBP, 2005.

Southworth, J., Munroe, D. et al. Land cover change and landscape fragmentation - comparing the utility of continuous and discrete analyses for a western Honduras region. **Agriculture Ecosystems & Environment**, v. 101, n. 2-3, p. 185-205, 2004.

Straatman, B., Hagen, A. et al. The Use of Cellular Automata for Spatial Modelling and Decision Support in Coastal Zones and Estuaria. **M. M. T. R. I. f. K. a. Systems. Maastricht**, The Netherlands: Maastricht University, 2001.

Takeyama, M.; Couclelis, H. Map dynamics: Integrating cellular automata and GIS through Geo-Algebra. **International Journal of Geographical Information Science**, v. 11, n. 1, p.73-91, 1997.

Turner II, B. L.; Ross, R. H. et al. **Relating land use and global land cover change:** IGDP report no. 24 - HDP report no. 5. Stockholm: Royal Swedish Academy of Sciences, 1993.

Turner II, B. L.; Skole, D. et al. L**and-Use and Land-Cover Change Science/Research Plan. I. R. N. a. H. R. N. 7**. Stockholm and Geneva, IGBP Secretariat, 1995.

Vanacker, V.; Linderman, M. et al. Impact of short-term rainfall fluctuation on interannual land cover change in sub-Saharan Africa. **Global Ecology and Biogeography**, v. 14, n. 2, p.123-135, 2005.

Vasconcelos, P. F.; Travassos da Rosa, A. P. et al.. Inadequate management of natural ecosystem in the Brazilian Amazon region results in the emergence and reemergence of arboviruses. **Cadernos de Saúde Pública**, v. 17, p. 155-164, 2001.

Veldkamp, A.; Fresco, L. O.. CLUE: a conceptual model to study the Conversion of Land Use and its Effects. **Ecological Modelling**, v. 85, p. 253-270, 1996.

Veldkamp, A.; E. F. Lambin. Predicting land-use change. **Agriculture Ecosystems & Environment**, v. 85, n. 1-3, p. 1-6, 2001.

Verburg, P., Schot, P. et al.. Land use change modeling: current practices and research priorities. **GeoJournal,** v. 61, n. 4, p. 309-324, 2004.

Verburg, P. H., Soepboer, W. et al.. Modeling the Spatial Dynamics of Regional Land Use: The CLUE-S Model. **Environmental Management**, v. 30, n. 3, p. 391-405, 2002.

Verburg, P. H., Veldkamp, A. et al.. A spatial explicit allocation procedure for modelling the pattern of land use change based upon actual land use. **Ecological modelling**, v. 116, p. 45–61, 2004.

von Neumann, J.. **Theory of self-reproducing automata**. Illinois: A.W. Burks, 1966.

Walker, R.. Theorizing land-cover and land-use change: The case of tropical deforestation. **International Regional Science Review**, v. 27, n. 3, p. 247-270, 2004.

Walker, R., Drzyzga, S. A.et al.. A behavioral model of landscape change in the Amazon Basin: The colonist case. **Ecological Applications**, v. 14, n. 4, p. S299-S312, 2004.

Werth, D.; Avissar, R.. The local and global effects of Amazon deforestation. **Journal of Geophysical Research-Atmospheres**, v. 107, n. D20, 2002.

Wesseling, C. G.; Karssenberg, D. et al. Integrating dynamic environmental models in GIS: the development of a Dynamic Modelling language. **Transactions in GIS**, v. 1, p. 40-48, 1996.

White, R.; Engelen, G. Cellular automata as the basis of integrated dynamic regional modelling. **Environment and Planning B: Planning and Design**, v. 24, p. 235-246, 1997.

White, R.; Engelen, G. et al. **Vulnerability assessment of low-lying coastal areas and small islands to climate change and sea level rise – Phase 2:** Case study St. Lucia. Kingston, Jamaica: United Nations Environment Programme - Caribbean Regional Co-ordinating Unit, 1998.

Wilson, E. O. Threats to Biodiversity. **Scientific American**, v. 261, n. 3, p. 108-&, 1989.

Wolfram, S. Cellular automata as models of complexity. **Nature**, v. 311, p. 419-424, 1984.

Wooldridge, M. J.; Jennings, N. R. Intelligent agents: Theory and practice. **Knowledge Engineering Review**, v. 10, n. 2, 1995.

Zeigler, B. P.; Kim, T. G. et al. **Theory of modeling and simulation**. Orlando, FL, USA: Academic Press, Inc, 2005.