MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-10288-TDI/907

AMBIENTE COMPUTACIONAL PARA MODELAGEM DINÂMICA ESPACIAL

Bianca Maria Pedrosa

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Gilberto Câmara, aprovada em 07 de agosto de 2003.

INPE
São José dos Campos
2004

681.3.06

PEDROSA, B. M.
Ambiente computacional para modelagem dinâmica espacial / B. M. Pedrosa. – São José dos Campos: INPE, 2003.
111p. – (INPE-10288-TDI/907).

1.Sistemas de Informação Geográfica (SIG). 2.Modelos dinâmicos. 3.Autômatos celulares. 4.Simulação ambiental. 5.Simulação de sistemas. I.Título.

Aprovada pela Banca Examinadora em cumprimento a requisito exigido para a obtenção do Título de **Doutora** em **Computação Aplicada.**

Dr. Antônio Miguel Vieira Monteiro

Presidente/INPE-SJCampos/SP

Dr. Gilberto Câmara Neto

Orientador/INPE-SJCampos/SP

Dr. João Ricardo de Freitas Oliveira

Membro da Banca/INPE-SJCampos/SP

Dr. Frederico Torres Fonseca

Membro da Banca
Convidado Penn State University, EUA

Dr. Marco Antonio Casanova

Membro da Banca
Convidado PUC, Rio de Janeiro/RJ

Candidato (a): Bianca Maria Pedrosa

São José dos Campos, 7 de agosto de 2003.

A meus pais,
Sylvio e Maria Izabel

AGRADECIMENTOS

Ao Dr. Gilberto Câmara, meu orientador, pelo suporte e inspiração.

Ao Dr. Frederico Fonseca, meu co-orientador na Penn State University, pela acolhida e amizade.

A Ricardo Cartaxo Modesto, pelas sugestões de implementação.

Aos colegas da equipe TerraLib, em especial Ana Paula Dutra de Aguiar e Lúbia Vinhas, pela preparação dos dados e suporte em TerraLib.

À CAPES, pela concessão da bolsa de estudos para estágio de doutorado no exterior.

À UNIMEP, pela concessão da bolsa de capacitação docente.

RESUMO

Esta tese apresenta TerraML, um ambiente computacional para modelagem dinâmica espacial para ser usado em aplicações ambientais. TerraML baseia-se num modelo celular de representação do espaço. TerraML utiliza o paradigma de autômatos híbridos para abstrair os elementos contínuos e discretos de um sistema dinâmico. Para capturar ações a distância, TerraML adota o conceito de vizinhança generalizada, implementada como uma matriz de proximidade, definida a partir de cálculos de distâncias entre as células ou a partir de redes de transporte.

A COMPUTATIONAL ENVIRONMENT FOR SPATIAL DYNAMIC MODELING

ABSTRACT

This thesis introduces TerraML, a computational environment for spatial dynamic modeling, to be used in environmental applications. TerraML represents space as a cellular model. In TerraML the discrete and continuous elements of a dynamic system are translated into control modes, flow and jump conditions, based on the hybrid automata theory. In order to capture action-at-a-distance TerraML adopts a generalized neighborhood concept, implemented as a proximity matrix, which can be defined based on the distance between cells or based on transportation networks.

SUMÁRIO

Pág.

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1
INTRODUÇÃO

Modelos dinâmicos espaciais realizam a simulação de processos do mundo real em que o estado de um objeto na superfície terrestre muda em resposta às suas forças direcionadoras (Burough 1998). Tais sistemas têm sido implementados através de modelos celulares, freqüentemente embutidos em estruturas computacionais baseadas em autômatos celulares.

Um autômato celular corresponde a uma grade regular em que cada célula pode assumir um valor discreto, o qual mudará em função dos estados das células em uma vizinhança adequadamente definida. Portanto, autômatos celulares são eficientes para representar sistemas em que a ordem global emerge de ações locais e descentralizadas. Entretanto, em Sistemas de Informações Geográficas, a ordem global depende também de ações à distância. A fim de superar esta limitação, e acomodar ações à distância, várias extensões à noção de autômato celular clássico têm sido propostas.

Além da capacidade de capturar ações à distância, um ambiente de modelagem dinâmica tem que ser capaz de abstrair os componentes discretos e contínuos de um sistema dinâmico. Os componentes discretos são os atributos das células que sofrem mudanças igualmente discretas, em virtude do seu espaço de estados finito. Por exemplo, uma célula de floresta pode mudar para desmatada e vice-versa (escala nominal). Os componentes contínuos de um modelo dinâmico são variáveis reais, que são atualizadas a partir de equacionamentos matemáticos. Nestes componentes, ao contrário dos componentes discretos, as mudanças de estado não são tão diretas. Por exemplo, para um modelo hidrológico é calculado a quantidade de água no solo por célula, em cada passo de tempo da simulação. Desta forma, as células são sistematicamente atualizadas. Entretanto, nem toda atualização da célula, implica em uma mudança de estado (escala intervalar).

Esta tese apresenta TerraML, um ambiente para modelagem dinâmica baseado num modelo celular de representação do espaço. Neste modelo as células possuem vários atributos e são organizadas sob a noção de um espaço celular (Batty 2000). Nesta perspectiva, o estado de uma célula é definido em função do estado dos seus

atributos. As regras de transição não se restringem a regras locais, podendo ser definidas em função das propriedades das células.

TerraML usa o paradigma de autômatos híbridos para abstrair os elementos contínuos e discretos de um sistema dinâmico. Neste paradigma, o sistema é modelado em modos de controle, condições de fluxo e de mudança, que capturam as dinâmicas contínua e discreta do sistema.

Para capturar ações à distancia, TerraML adota o conceito de vizinhança generalizada, implementada como uma matriz de proximidade, que pode ser definida a partir de cálculos das distâncias entre as células ou a partir de redes de transportes. Com esta abordagem uma célula pode ter qualquer forma e tamanho de vizinhança, flexibilizando assim a estacionariedade das vizinhanças presente nos autômatos celulares clássicos.

Para testar a aplicabilidade de TerraML, foi desenvolvida uma aplicação para simular a mudança na cobertura do solo de um área do estado de Rondônia, onde o desmatamento é fortemente determinado pela acessibilidade. Os resultados obtidos com esta aplicação não são conclusivos, mas são indicativos da capacidade do ambiente computacional. O ambiente trabalha com uma lógica simplificada, que pode ser refinada com a ampliação do número de funções disponíveis, a serem implementadas trabalhos futuros.

1.1 - Objetivos

TerraML foi desenvolvida com o objetivo de diminuir algumas das limitações existentes nos ambientes computacionais para modelagem dinâmica disponíveis atualmente. Algumas destas limitações são:

1) Especialização dos Sistemas / limitação das aplicações
   Sistemas de modelagem dinâmica geralmente são especializados, isto é, são dedicados a uma classe de aplicações e não podem ser facilmente adaptados a outras situações de modelagem. Um dos objetivos da TerraML é ser um ambiente de simulação dinâmica de propósito geral, suportando uma ampla gama de aplicações. Aplicações potenciais do ambiente TerraML são processos de desmatamento (Lambin 1994), processos de expansão urbana

(Couclelis 1997), (White and Engelen 1997) e de processos ecológicos (Soares Filho 1998).

2) Nível de Acoplamento Fraco

   A maioria dos sistemas de simulação são sistemas fracamente acoplados, isto é, demandam conversões e um grande trabalho de pré-processamento dos dados. Um dos objetivos de TerraML é a integração plena com o Sistemas de Gerenciamento de Banco de Dados de uso geral.

3) Sistemas fechados/ soluções proprietárias

   O alto nível de especialização dos sistemas de modelagem dinâmica leva estes sistemas a apresentarem um alto custo. Além disto, estes sistemas são fechados, não podendo ser modificados ou estendidos.

   TerraML é parte da biblioteca espacial TerraLib (Câmara, Souza et al. 2000). TerraLib está sendo desenvolvida sobre os fundamentos do software livre, ou seja, a distribuição do software é gratuita e todo código pode ser copiado e modificado.

4) Restrição de Modelos

   Muitos sistemas de modelagem restringem-se a um modelo matemático, excluindo a possibilidade do uso combinado deste modelo com outros. Um dos objetivos deste trabalho é trabalhar com a visão de Lambin (1994) de que os diferentes modelos matemáticos não são excludentes, mas sim complementares. Nesta perspectiva, o sistema deve suportar o uso combinado de diferentes modelos matemáticos e permitir a interação de componentes discretos e contínuos, independente da solução computacional implementada.

1.2 - Organização da Tese

No Capítulo 2, **Modelagem Dinâmica e GIS**, é apresentada uma revisão bibliográfica sobre modelagem dinâmica, focalizando os princípios básicos e requisitos necessários para representar os principais componentes de um modelo espacial dinâmico. Neste capítulo também, são descritos dois ambientes computacionais para Modelagem dinâmica espacial, o PCRaster e o Modelo Multi-Escala do RIKS.

No Capítulo 3, **Ambiente Computacional para Modelagem Dinâmica Espacial**, é apresentado o ambiente computacional para modelagem ambiental desenvolvido nesta tese. O capítulo está organizado em duas partes: aspectos conceituais, que apresenta os fundamentos teóricos sobre os quais este ambiente foi projetado, e aspectos de implementação, que apresenta as estruturas de dados e tecnologia utilizada para implementar o ambiente computacional para dar suporte à linguagem TerraML

No Capítulo 4, **TerraML - Formalização da Linguagem**, são apresentados os principais elementos desta linguagem. Este capítulo está organizado em arquitetura, sintaxe e funcionalidade da linguagem.

No Capítulo 5, **Uma Aplicação em Mudança na Cobertura do Solo,** apresentamos uma visão geral da TerraML, através de um exemplo de um processo de mudança no uso e cobertura do solo relativo à evolução do desmatamento na Amazônia.

No Capítulo 6, **Conclusões e Futuros Trabalhos,** são apresentadas algumas considerações finais sobre o trabalho e sugestões de temas de pesquisa para futuros trabalhos em modelagem dinâmica.

CAPÍTULO 2
MODELAGEM DINÂMICA E GIS

2.1 - Requisitos para Modelagem Dinâmica em GIS

A atual geração de Sistemas de Informações Geográficas (GIS) configura uma tecnologia estabelecida para armazenar, organizar, recuperar e modificar informações sobre a distribuição espacial de recursos naturais, dados geo-demográficos, redes de utilidade pública e outros tipos de dados localizados na superfície da terra. Nesta área, um dos principais desafios atuais é transformar estes sistemas, essencialmente estáticos, em ferramentas capazes de prover representações realistas de processos espaço-temporais. A modelagem de grande quantidade de processos físicos, em aplicações como geomorfologia, estudos climáticos, dinâmica populacional, e impacto ambiental, requer que os GIS tenham capacidade de representar os tipos de processos dinâmicos encontrados em estudos de sistemas físicos e sócio-econômicos.

Neste contexto, a modelagem dinâmica (Burrough, 1998) procura transcender as limitações atuais da tecnologia de Geoprocessamento, fortemente baseada numa visão estática, bidimensional do mundo. O objetivo dos modelos dinâmicos em GIS é realizar a simulação numérica de processos dependentes do tempo, como nos modelos hidrológicos, que simulam o fluxo e transporte de água. Na definição de Burrough, “um modelo espacial dinâmico é uma representação matemática de um processo do mundo real em que uma localização na superfície terrestre muda em resposta a variações em suas forças direcionadoras”.

Tipicamente, GIS são desenvolvidos a partir de suposições pré-estabelecidas quanto à homogeneidade, uniformidade e universalidade das propriedades de seus principais componentes, que incluem o espaço e as relações espaciais, o tempo e o modelo matemático que descreve o fenômeno. Entretanto, para modelar processos dinâmicos em GIS com o nível necessário de realismo, estas suposições rígidas têm que ser flexibilizadas de tal forma que o sistema seja capaz de representar (Couclelis, 1997):

- O espaço como uma entidade não homogênea tanto nas suas propriedades quanto na sua estrutura.

- As vizinhanças como relações não estacionárias.
- As regras de transição como regras não universais.
- A variação do tempo como um processo regular ou irregular.
- O sistema como um ambiente aberto a influências externas.

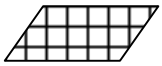
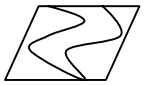
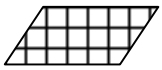
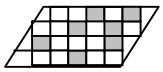
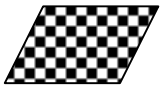
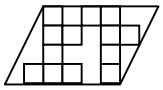
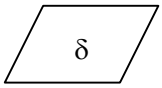
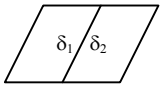
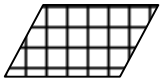
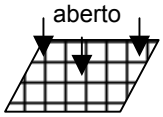
Espaço	Estrutura	regular 	Irregular 
	Propriedades	uniforme 	não uniforme 
Vizinhança		estacionária 	não estacionária 
Função de Transição		universal 	não universal 
Tempo		regularidade	Irregularidade
Sistema		fechado 	aberto 

FIGURA 2.1 – Requisitos para modelagem dinâmica em GIS.
FONTE: Couclelis (1997).

Na Figura 2.1 estão representados os requisitos mencionados anteriormente. A regularidade do espaço diz respeito à forma como ele é distribuído, e pode ser regular, isto é, dividido em parte iguais, ou irregular, distribuído de forma diferenciada. As vizinhanças, que geralmente são concebidas como tendo a mesma configuração para todo ponto no espaço, devem superar esta estacionaridade e podem ser representadas com diferentes configurações em diferentes pontos do espaço. Por exemplo, em determinado ponto uma célula pode ter vizinhança 4, e em outro ponto, vizinhança 8. O sistema deve permitir que mais de uma função de transição possa ser aplicada, permitir que o tempo seja representado em intervalos variáveis (meses, anos) e suportar a inclusão de variáveis externas.

Para implementar sistemas espaciais dinâmicos com as características mencionadas acima, alguns princípios básicos relativos aos principais elementos destes sistemas

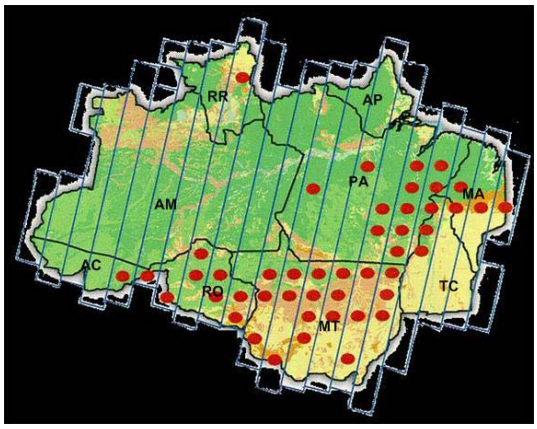
devem ser considerados. Entre estes elementos destaca-se a questão da representação do espaço e do tempo, o modelo dinâmico a ser utilizado para a representação do fenômeno espacial e a abordagem computacional para implementar estes princípios de forma integrada e consistente. Nas seções seguintes, discutiremos cada um destes elementos.

2.1.1 -O Espaço

O espaço é o conceito chave na geografia e, por extensão, na Ciência da Informação Espacial. Tradicionalmente, os geógrafos fazem uma distinção entre os conceitos de espaço absoluto e espaço relativo.

"Espaço absoluto, também chamado Cartesiano ou Newtoniano, é um container de coisas e eventos, uma estrutura para localizar pontos, trajetórias e objetos. Espaço relativo, ou Leibnitziano, é o espaço constituído pelas relações espaciais entre coisas e eventos" (Couclelis, 1997).

(Santos, 1996) refere-se à distinção entre espaço absoluto e espaço relativo como o "espaço dos fixos" e o "espaço dos fluxos". Em termos de representações computacionais pode-se, de forma aproximada, traduzir estes conceitos como a distinção entre as representações associadas a recobrimentos planares (mapas de polígonos e matrizes) e representações associadas a conectividade (grafos). Um caso típico de medida realizada no espaço absoluto são os índices de auto-correlação espacial. Neste caso, um dos instrumentos básicos é a matriz de proximidade espacial, cujo cálculo usualmente é feito em função de distância euclidiana entre objetos ou da existência de uma fronteira entre eles. Na Figura 2.2 está representado um mapa temático e sua respectiva matriz de proximidade definida com base nas fronteiras existentes entre os objetos, no caso, estados da federação.



	AC	AM	AP	MA	MT	PA	RO	RR	TO
AC		1					1		
AM	1				1	1	1	1	
AP						1			
MA						1			1
MT		1				1	1		1
PA		1	1	1	1	1		1	1
RO	1	1			1				
RR		1				1			
TO				1	1	1			

FIGURA 2.2 – Um mapa poligonal e sua matriz de proximidade.

Em muitos fenômenos geográficos, os objetos estabelecem relações entre si que independem das relações espaciais típicas, como as relações topológicas, direcionais e de distância. Estes fenômenos geralmente incluem relações como fluxo de pessoas ou materiais, conexões funcionais de influência, comunicação e acessibilidade, entre outras (Couclelis, 1999). Um exemplo de fenômeno em que a dimensão espacial requer o conceito de espaço relativo é o caso de fluxo de pessoas pela rede de transporte metroviário de uma cidade. O fluxo de pessoas a partir de uma mesma origem tem diferentes destinos, como mostrado na Figura 2.3, e a relação entre a origem e o destino, é estabelecida com base em relações de conectividade e acessibilidade.

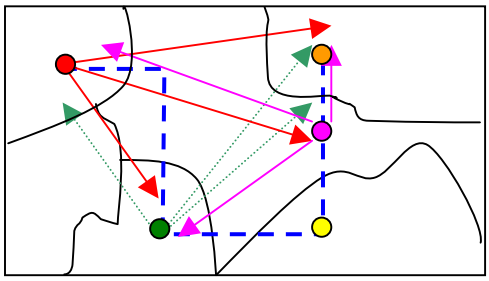
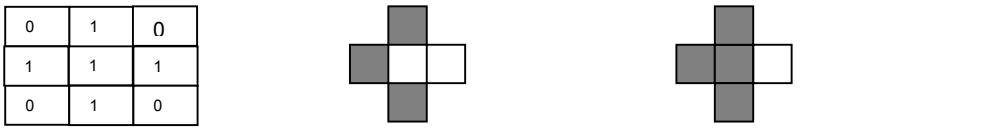


FIGURA 2.3 – Mapa do fluxo de pessoas em uma rede de transporte.

Couclelis (1997) propõe a idéia de espaço próximo como uma extensão dos conceitos de espaço absoluto e relativo. No espaço próximo o conceito chave é a vizinhança associada à noção de proximidade, que conduzem também ao conceito de proximidade funcional ou influência. O conceito de vizinhança é facilmente visualizado em representações matriciais do espaço. Algumas operações espaciais disponíveis

em GIS como filtros espaciais, por exemplo, utilizam a noção de espaço próximo de forma limitada. No filtro espacial, o estado de uma célula (um pixel de uma imagem) é modificado com base nos estados das demais células em sua vizinhança, definida através de uma máscara. A seguir é apresentado um exemplo de filtro espacial e o estado de uma célula qualquer, antes (Figura 2.4b) e após (Figura 2.4c) a aplicação do filtro espacial.



a) Máscara b) estado da célula central antes do filtro c)estado da célula central após o filtro

FIGURA 2.4 – Exemplo de filtro espacial.

A abstração fundamental na maior parte dos GIS atuais é o conceito de mapa, fortemente relacionado com noções cartográficas e, portanto, do espaço absoluto. Em processos dinâmicos a noção de espaço relativo e próximo são fundamentais para estabelecer e representar fluxos e conexões entre entidades do sistema.

2.1.2 O - Tempo

Apesar de ser reconhecida a necessidade de incorporar a dimensão temporal em muitos processos ambientais, a representação do tempo em sistemas de informações geográficas não passou de um estágio embrionário (Parent et al., 1999; Zipf e Krüger, 2001). Isto se deve (1) ao paradigma cartográfico sobre o qual GIS foram construídos, (2) a ênfase dada em soluções orientadas a implementação, e (3) a ausência de uma teoria espaço-temporal (Peuquet, 2001). A maioria das implementações de aspectos temporais em GIS tem sido limitada a extensões de sistemas espaciais para acomodar frágeis conceitos de tempo, ignorando (1) a semântica dos processos espaço-temporais e (2) os aspectos subjacentes da mudança (Hornsby e Egenhofer, 1997).

Conceitualmente, pode-se representar o tempo através de diferentes estruturas, definidas, principalmente, com base em três aspectos da representação temporal: granularidade, variação e ordem no tempo (Figura 2.5).

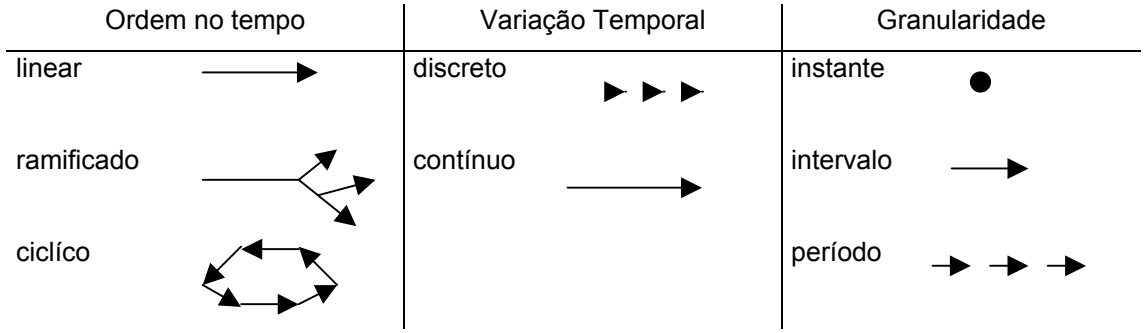


FIGURA 2.5 - Estruturas temporais.
FONTE: Worboys (1998).

A ordem temporal refere-se ao modo como o tempo flui. Neste caso, pode-se assumir que o tempo flui de forma linear, ramificada ou cíclica. No tempo linear considera-se que o tempo flui seqüencialmente, ou seja, existe uma ordem de precedência entre os pontos no tempo, de tal forma que cada ponto tenha apenas um sucessor e um antecessor. No tempo ramificado, múltiplos pontos podem ser os sucessores ou antecessores imediatos de um mesmo ponto. O tempo cíclico é utilizado para modelar eventos e processos recorrentes (Edelweiss e Oliveira, 1994).

Com relação à variação temporal, duas possibilidades podem ser consideradas: tempo contínuo e discreto. Uma variável temporal contínua é usada em processos que demandam medidas de tempo com níveis arbitrários de precisão. Por exemplo, a expansão da área de desmatamento de uma floresta entre dois instantes de tempo medidos pode ser interpolada. Uma variável temporal discreta é usada quando o tempo é medido em certos pontos ou intervalos e a variação é descontínua entre estes pontos. Uma delimitação de lotes de um cadastro imobiliário pode ocupar uma posição num instante t e outra num instante t', mas não faz sentido dizer que a delimitação ocupou alguma posição intermediária entre t e t'.

Associado ao conceito de variação temporal discreta, existe o conceito de *Chronon*. Um *chronon* é a menor duração de tempo suportada por um sistema e pode variar em diferentes aplicações (Edelweiss e Oliveira, 1994).

A granularidade temporal de um sistema está diretamente relacionada com a duração de um *chronon*. As diferentes granularidades de um sistema temporal conduzem à definição de instante e intervalo de tempo. Um instante de tempo representa um ponto particular no tempo, um intervalo é o tempo decorrido entre dois instantes e um período consiste de uma seqüência de intervalos de tempo .

Em sistemas computacionais, representa-se o tempo em pelo menos duas dimensões:

- tempo válido (valid *time*) - corresponde ao tempo em que um evento ocorre no domínio da aplicação.
- tempo de transação (*transaction time*) – corresponde ao tempo em que transações acontecem dentro do sistema de informação (Worboys, 1995).

Adicionalmente, existe o conceito de "tempo definido pelo usuário", consistindo de propriedades definidas explicitamente pelos usuários em um domínio temporal e manipuladas pelo programa de aplicação (Edelweiss e Oliveira, 1994).

A incorporação da dimensão temporal em um sistema de informação não se restringe apenas à questão da representação do tempo, mas inclui também questões relativas a sua recuperação. Um GIS temporal deve ser capaz de recuperar informações através de consultas definidas sobre critérios temporais, como por exemplo:

- Quais rodovias do Brasil foram recuperadas a **partir de** 1980 e **agora** permitem uma velocidade superior a 100km/h?
- Qual rio teve a maior taxa de poluição **entre** 1970 e 1985?
- Quais as cidades em que a cobertura vegetal aumentou em pelo menos 5% **durante** os últimos 5 anos?

A representação do tempo em modelos dinâmicos não se limita a uma simples questão de estender os GIS para incorporar conceitos de banco de dados temporais. Na dimensão temporal, assim como na dimensão espacial e do modelo, a dicotomia entre contínuo e discreto é uma questão desafiante. Eventos tais como relâmpagos e erupções vulcânicas são discretos tanto no domínio temporal quanto no espacial, enquanto temperatura e precipitação são processos espaço-temporais contínuos (Peuquet, 2001). Outro conceito forte em sistemas temporais refere-se à dinâmica de atualização dos objetos, que pode ser síncrona ou assíncrona. Em sistemas síncronos todos os elementos do sistema são atualizados simultaneamente (Sipper, 1999). Já em sistemas assíncronos, os elementos não são atualizados em intervalos de tempo regulares.

2.1.3 -Modelos

Modelos espaciais dinâmicos descrevem a evolução de padrões espaciais de um sistema ao longo do tempo. Segundo Lambin(1994) o modelo de um fenômeno deve responder às seguintes questões:

- **Quais** variáveis ambientais e culturais contribuem para explicar o fenômeno, e quais são os processos ecológicos e sócio-econômicos existentes por trás do fenômeno?
- **Como** o processo evolui?
- **Onde** ocorrem os fenômenos?

Estas questões chaves podem ser identificadas como as clássicas “Porquê”, “Quando” e “Onde”. Um modelo que responde a estas questões é capaz de descrever quantitativamente um fenômeno e prever sua evolução, integrando suas escalas temporal e espacial.

Um modelo é constituído de pelo menos três elementos variáveis, relacionamentos e processos. Ao se construir um modelo, dependendo do objetivo, pode-se dar ênfase a um ou outro destes elementos. Nesta visão, os modelos podem ser classificados em empíricos e sistêmicos, Figura 2.6. Modelos empíricos focalizam os relacionamentos entre as variáveis do modelo, a partir da suposição de que os relacionamentos observados no passado continuarão no futuro. Modelos sistêmicos são descrições matemáticas de processos complexos que interagem entre si, enfatizando as interações entre todos os componentes de um sistema (Lambin, 1994).

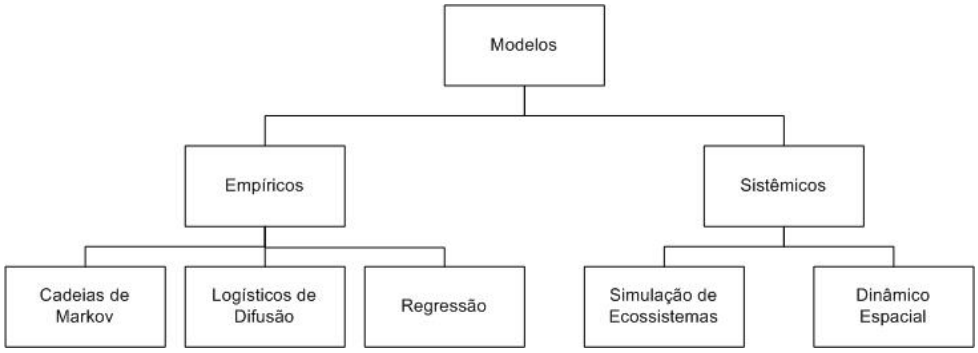


FIGURA 2.6 – Tipos de modelos.
FONTE: adaptada de Lambin (1994).

2.1.3.1 - Modelos Empíricos

Os modelos empíricos, em sua dimensão procedimental, possuem três componentes chaves: uma configuração inicial, uma função de mudança e uma configuração de saída. A configuração inicial de um modelo dinâmico pode ser obtida através de dados históricos do fenômeno em estudo, chamados de séries temporais. Neste caso, equações diferenciais (totais ou parciais) que incluem pelo menos um termo derivado no tempo podem ser utilizadas para representar o modelo e o processo é classificado como determinístico. Quando variáveis aleatórias são utilizadas para explicar um sistema, o processo é classificado como estocástico-probabilístico.

Modelos empíricos são caracterizados pela simplicidade dos modelos matemáticos empregados e pelo número reduzido de variáveis envolvidas. Estes modelos são eficientes em fazer predições, embora apresentem limitações em abordar a evolução espacial e identificar os aspectos causais do sistema. A seguir, serão apresentados três modelos empíricos: cadeias de markov, modelos logísticos de difusão e modelos de regressão.

2.1.3.1.1 - Cadeias de Markov

Cadeias de Markov são modelos matemáticos para descrever processos estocásticos e podem ser denotadas por:

$$\Pi(t+1) = P^n . \Pi(t)$$

onde $\Pi(t)$ é o estado do sistema no tempo t, $\Pi(t+1)$ é o estado do sistema após o instante t+1 e P^n são os estados passíveis de acontecer, que são representados em matrizes de possibilidades de transição. Essas matrizes de transição representam a possibilidade de um determinado estado i permanecer o mesmo ou mudar para o estado j durante o instante de tempo t->t+1. As probabilidades de transição são usualmente derivadas de amostras relativas a um certo instante de tempo. Cadeias de Markov de 1ª ordem assumem que o estado futuro do sistema depende apenas do seu estado presente e das possibilidades de transição, sendo independente da trajetória que o levou àquele estado (estados em um tempo t-1). Este modelo não

ignora o passado, mas assume que toda a informação do passado está concentrada no presente estado do sistema. Desta forma, as interações são instantâneas, sendo irrelevante o tempo de permanência das variáveis em cada estado (Soares Filho, 1998).

Outra característica das cadeias de Markov é que as probabilidades de transição não mudam com o tempo, o que o caracteriza como um processo estacionário.

As principais vantagens das cadeias de Markov são a simplicidade operacional e matemática do modelo aliadas à facilidade com que podem ser aplicadas a dados provenientes de sensoriamento remoto e implementadas em GIS. Outra grande vantagem é o fato de não necessitar de grande quantidade de dados antigos para prever o futuro.

As principais limitações das cadeias de Markov incluem o fato do modelo não explicar o fenômeno (Porquê) e ser limitado na resposta espacial (Onde), entretanto o modelo pode fazer predições (Quando) desde que os processos sejam estacionários. Além disto, o modelo não suporta de imediato a inclusão de variáveis exógenas como variáveis sócio-econômicas ou outras forças direcionadoras, embora esta limitação possa ser superada. Em (Lambin, 1994) são apresentadas várias abordagens para superar as principais limitações de cadeias de Markov em modelagem dinâmica.

2.1.3.1.2 - Modelos logísticos de Difusão

Modelos logísticos são utilizados para descrever matematicamente fenômenos em que as variáveis inicialmente apresentam variações em um ritmo lento, depois o ritmo de variações se intensifica, voltando a reduzir-se até que o nível de saturação seja atingido. Este modelo leva em conta as interações temporais entre as variáveis do sistema, podendo ser expresso por:

$$\frac{dP}{dt} = r\,P\left[\frac{(U - P)}{U}\right]$$

onde P é a variável de um fenômeno de crescimento ao longo do tempo t, como aumento da população, por exemplo: r é a taxa de crescimento e U uma função de crescimento (Lambin, 1994).

Dentre os modelos baseados em funções logísticas destacam-se os modelos de difusão. Tais modelos enfatizam a velocidade do processo e permitem a inclusão de variáveis relacionadas às causas do fenômeno.

Os principais elementos de um modelo espacial de difusão são (Soares Filho, 1998):

- meio ambiente (isotrópico ou heterogêneo)
- tempo (contínuo ou discretizado)
- item a ser difundido (material, pessoas, informação, doença)
- locais de origem
- locais de destino
- caminhos a serem percorridos

Estes elementos interagem entre si através de um mecanismo em que se pode identificar quatro estágios:

- Estágio inicial – neste estágio tem início o processo de difusão.
- Estágio de difusão – tem início o processo de espalhamento
- Estágio de condensação – diminui o ritmo do espalhamento.
- Estágio de saturação – ocorre a desaceleração ou encerramento do processo de difusão.

O processo de espalhamento em modelos de difusão pode se dar por expansão ou realocação. Nos modelos de difusão por expansão a informação ou material se espalha de uma região para outra, permanecendo na região original. Nos modelos de difusão por realocação os objetos se movem para novas regiões, abandonando as área originais (Soares Filho, 1998).

Modelos de difusão não explicam as causas de um fenômeno, embora possam integrar variáveis ecológicas e sócio-econômicas. Sua maior contribuição está na predição do comportamento futuro do fenômeno. Quanto à dimensão espacial, o modelo em si não a incorpora, mas ela pode ser introduzida através da integração deste modelo com um GIS (Lambin, 1994).

2.1.3.2.3 - Modelos de Regressão Linear

O objetivo dos modelos de regressão é estabelecer relações estatísticas entre um fenômeno em estudo e as variáveis independentes envolvidas, chamadas forças direcionadoras, que exercem influência sobre ele. Sendo assim, o modelo suporta a

inclusão de variáveis exógenas como as sócio-econômicas. Isto contribui para o entendimento do fenômeno em estudo, mas é insuficiente para explicá-lo, pois a identificação de um relacionamento estatístico entre duas variáveis por si só não estabelece um relacionamento causal entre elas. Por exemplo, pode-se identificar através de um modelo de regressão que o crescimento populacional tem relação com o crescimento do desmatamento de uma determinada região, entretanto, o modelo de regressão não explica os mecanismos que ligam estas variáveis (Lambin, 1994).

Matematicamente, o modelo estabelece um relacionamento linear entre as variáveis dependentes e independentes através da expressão:

$$y = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_i x_i + E$$

onde:

y = mudança ocorrida em um determinado tempo
$\mathbf{x_i}$ = variáveis independentes (forças direcionadoras)
$\mathbf{a_i}$ = Coeficientes de regressão dos relacionamentos
E = Componente de erro

Em modelos de regressão a dimensão temporal é considerada, mas a distribuição espacial do fenômeno não é abordada, limitação esta que pode ser superada se o modelo for combinado com GIS. Outra limitação deste modelo é que ele se aplica apenas a processos estacionários (Lambin, 1994).

Um exemplo de modelo de regressão é o implementado por (Reis e Margulis, 1991) para modelar o desmatamento da Amazônia em função da densidade espacial das atividades econômicas da região. Neste modelo, num primeiro estágio, áreas desmatadas são relacionadas com a densidade populacional, áreas cultivadas, distância de centros urbanos e proximidade de rodovias, entre outras variáveis. Num segundo estágio, o modelo relaciona o crescimento de determinadas atividades (colonização, cultivo, pecuária) entre 1980 e 1985 com a densidade destas atividades em 1980, obtendo assim o padrão de crescimento espacial de cada atividade. Então, partindo da suposição de que este padrão espacial de crescimento irá se manter no futuro, o modelo faz projeções sobre a tendência de desmatamento para o período de 1985-2000 (Lambin, 1994) .

2.1.3.2 - Modelos Sistêmicos

Modelos sistêmicos procuram descrever o sistema como um todo, isto é, tentam representar as interações entre todos os seus componentes. Uma característica chave destes modelos é a eficiência com que abordam a dimensão espacial, implementando conceitos como as relações de vizinhança e suportando o uso combinado de múltiplas escalas. A seguir, descreveremos as características gerais de duas classes de modelos sistêmicos: os modelos de simulação de ecossistemas e os modelos de simulação dinâmica espacial.

2.1.3.2.1 - Modelos de Simulação de Ecossistemas

Modelos de Ecossistemas são projetados para imitar o comportamento de um sistema, enfatizando as interações entre todos os seus componentes. Estes modelos são baseados na composição de ecossistemas complexos em um número de equações diferenciais (Lambin, 1994). A construção de um modelo de simulação requer que os principais aspectos que afetam o fenômeno estejam bem integrados, que seus relacionamentos funcionais estejam bem representados e que o modelo possa predizer os impactos ecológicos e econômicos das mudanças ao longo do tempo.

Estes modelos são adequados para representar processos não estacionários, mas apresentam dificuldades para sua expressão espacial , pois tratam o espaço como uma entidade homogênea (Lambin, 1994).

2.1.3.2.2 - Modelos de Simulação Dinâmica Espacial

Modelos de Simulação Dinâmica Espacial baseiam-se em modelos de ecossistemas com extensões para acomodar a heterogeneidade espacial e processos humanos de tomada de decisão.

Uma abordagem para desenvolver modelos de simulação dinâmica espacial é representar o espaço como uma matriz de células e aplicar as equações matemáticas a cada uma das células da matriz, simultaneamente. Cada célula do modelo está conectada com suas células vizinhas, de tal forma que é possível estabelecer um fluxo

entre células adjacentes. Isto simplifica sobremaneira o mecanismo de predições do sistema porque, por exemplo, se uma célula tem três vizinhos com estado x, é altamente provável que o estado desta célula venha a ser x também. Entretanto, este raciocínio simplista pode ser aperfeiçoado através do uso de regras de transição. Outro aperfeiçoamento desse modelo é a possibilidade de incorporar processos de tomada de decisões. Modelos que incorporam este mecanismo são chamados modelos baseados em regras. As regras de tomada de decisão são representadas através de abstrações muito semelhantes àquelas que ocorrem na mente humana.

Um exemplo de modelo com as funcionalidades mencionadas acima é o implementado no DELTA (Dynamic Ecological Land Tenure Analysis), um sistema desenvolvido para integrar aspectos sócio-econômicos da colonização amazônica e aspectos ecológicos do desmatamento e da liberação de carbono na atmosfera no Estado de Rondônia.

O DELTA utiliza três submodelos integrados que simulam, respectivamente, a difusão da colonização, mudança do uso do solo e liberação de carbono. Os submodelos são examinados em diferentes escalas, o que caracteriza o modelo como multi-escala. Além disto, o modelo é considerado “a playing game tool”, pois não se restringe a fazer predições, mas sim a servir como instrumento para responder “what if questions” (Lambin, 1994).

2.1.3.3 - Resumo dos Modelos

Para finalizar, um resumo das características chaves de cada tipo de modelo, segundo o potencial de cada um deles para responder as perguntas porque, quando e onde é apresentado na Tabela 2.1.

Cadeias de Markov, modelos logísticos de Difusão e Regressão são eficientes em modelar processos estacionários. Estes modelos utilizam equações matemáticas simples e requerem poucos dados, além de serem compatíveis com o formato de dados oriundos de fontes de sensoriamento remoto e, como conseqüência, facilmente implementados em GIS.

Modelos Sistêmicos são considerados modelos exploratórios porque fornecem condições para que várias simulações possam ser investigadas a partir de diferentes cenários. Entretanto, estes modelos requerem um substancial conhecimento do

fenômeno em estudo e acabam por se tornar sistemas altamente especializados, não podendo ser aplicados a outras classes de fenômenos.

TABELA 2.1 Resumo dos tipos de modelos.

Modelo	Porquê	Quando	Onde
Cadeias de Markov	não pode explicar a razão de um fenômeno por ser processo estocástico e não suportar a inclusão de variáveis exógenas	pode predizer a evolução de processos estacionários	pode predizer distribuições espaciais de elementos do modelo se for combinado com GIS
Logístico de Difusão	permite a inclusão de poucas variáveis exógenas, entretanto é um modelo descritivo, não suportando investigações exploratórias	suporta a dimensão temporal, podendo predizer a evolução de processos não estacionários	pode predizer distribuições espaciais de elementos do modelo se for combinado com GIS
Regressão Linear	contribui para identificar forças direcionadoras, entretanto são modelos descritivos, não sendo capaz de estabelecer relações causais entre as variáveis	pode predizer a evolução de processos estacionários	não são modelos espaciais, entretanto podem ser combinados com GIS
Simulação de Ecossistemas	modelo exploratório que requer descrições funcionais dos sistemas ecológicos	pode formular cenários de mudanças futuras no uso do solo, baseado nos parâmetros do modelo	apresenta dificuldades na representação espacial
Simulação Espacial Dinâmica	requer modelos funcionais espacialmente definidos	pode predizer mudanças temporais no uso do solo, baseado nos parâmetros do modelo	pode predizer evolução de padrões espaciais em processos determinísticos

Diferentes modelos servem a diferentes propósitos, logo eles não são excludentes, mas sim complementares. Nesta perspectiva, Lambin (1994) sugere que, ao se construir um modelo, deve-se fazê-lo de forma gradual, começando por Cadeias de Markov, que são os mais simples, e ir incorporando novos elementos (variáveis exógenas) e funções (determinísticas) ao projeto.

A dimensão espacial deve ser também introduzida de forma gradual, começando com as relações espaciais mais elementares como as de vizinhança, refinando continuamente, de forma a contemplar a noção de espaço relativo e suporte a representações em múltiplas escalas.

Modelos espaciais dinâmicos construídos com esta visão de projeto devem ser capazes de representar de forma realista os fenômenos dinâmicos encontrados na natureza, superando as limitações dos modelos atuais, baseados em concepções limitadas quanto às representações do espaço, do tempo e dos processos.

2.2 - Ambientes Computacionais para Modelagem Dinâmica

Os GIS disponíveis atualmente foram implementados segundo diferentes paradigmas computacionais. A ciência de informação geográfica faz uso intenso das ferramentas e tecnologias computacionais disponíveis, além de impulsionar o desenvolvimento de novas abordagens para lidar com a natureza complexa dos dados e fenômenos espaciais. Esta forte interação entre a Ciência da Computação e a Ciência da Informação Espacial teve como conseqüência o surgimento do termo GeoComputação para significar o uso em larga-escala de paradigmas computacionais como ferramenta para pesquisas geográficas (Openshaw, 2000).

Na seção anterior, foram apresentados os princípios básicos relativos aos principais componentes de um modelo espacial dinâmico. Nesta seção, dois ambientes de simulação dinâmica, PCRaster e Modelo Multi-Escala/RIKS, serão analisados segundo aspectos computacionais como a arquitetura de software, estruturas de dados e de controle, funcionalidade e interface.

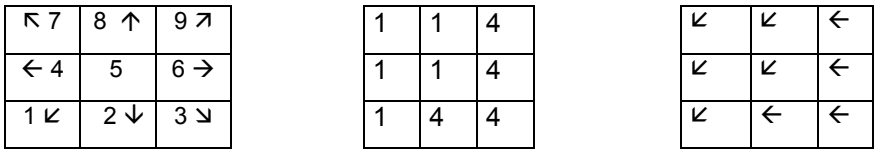
2.2.1 - PCRaster

O PCRaster é um software para modelagem dinâmica de processos físicos. Em processos físicos, a modelagem dinâmica descreve ou simula a distribuição, fluxo e transporte de material ou energia no solo (Van Deursen, 1995).

Para simular o transporte de material são necessárias estruturas de dados que suportem a noção de direção e relações de resistência, acúmulo de material e movimento. Nesta visão, a complexidade dos modelos dinâmicos depende da dimensão em que tais modelos operam, 2D ou 3D, e dos modelos matemáticos que utiliza .

O PCRaster trabalha com dados espaciais no modo matricial (*raster).* Para modelar um processo de fluxo/distribuição de material neste software, é utilizada uma estrutura de dados chamada *Local Drain Direction* (LDD). A estrutura de dados LDD consiste em uma matriz, equivalente à dos mapas da área de estudo, com direções para o fluxo de material. Essas direções consistem em números como os do teclado numérico do

computador (Figura 2.7 a). Desta forma, um LDD da forma apresentada na Figura 2.7 b, levaria ao fluxo de material segundo a representação da Figura 2.7 c.



a)Teclado numérico b) Representação computacional c) Representação conceitual

FIGURA 2.7 - Redes LDD.

Além do LDD e dos mapas no modo raster, o PCRaster utiliza séries temporais (arquivos tss), que são arquivos ASCII, contendo dados armazenados de forma tabular. Os dados fornecidos nas séries temporais são utilizados para calcular o novo estado das células.

O PCRaster oferece uma interface de programação baseada em scripts, através dos quais o usuário pode fazer uso extensivo das funções e operadores oferecidos na biblioteca de funções, desenvolvida em Linguagem C.

Para descrever o modo como o PCRaster é operado, utilizaremos como exemplo um caso de escoamento de água da chuva em uma bacia. Para modelar este processo é necessário fornecer como entradas para o sistema o Modelo Numérico do Terreno (MNT) e as séries temporais com os dados de precipitação pluviométrica. A partir do MNT (Figura 2.8 a) é gerada a rede LDD (Figura 2.8 b), que é a rede de drenagem por onde a água excedente flui. A água excedente é toda a água que não foi infiltrada, por já ter excedido a capacidade de infiltração da célula. Para determinar o padrão espacial do processo de infiltração, um mapa de solos da área em estudo (Figura 2.8 d) tem que ser fornecido. A partir destes dados, o programa é executado e gera um conjunto de mapas resultantes (Figura 2.8 e).

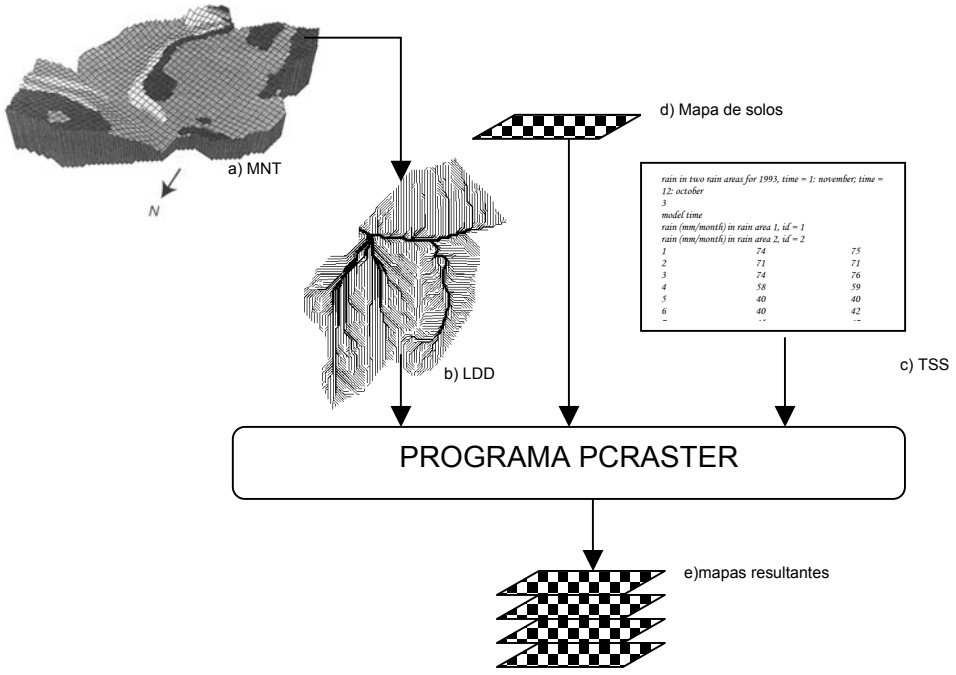


FIGURA 2.8 – Esquema simplificado das entradas e saídas do PCRaster.

Para modelagem dinâmica, o PCRaster oferece um conjunto de funções para o fluxo de material, tais como:

- fluxo acumulado (`accuflux/state`)- calcula o novo estado dos atributos de uma célula, somando o valor original da célula mais a soma acumulada de todos as células cujo fluxo passa por esta célula (*upstream cell)*;
- capacidade de transporte de uma célula (`accucapacity-flux/state`) - limita o fluxo de célula para célula a um atributo de capacidade de transporte fornecido em valores absolutos;
- fração de transporte (`accufractionflux/state`) - limita o fluxo sobre a rede a um parâmetro que controla a proporção de material que pode fluir em cada célula.
- valor limite (`accuthresholdflux/state`) – modifica o acúmulo de fluxo sobre a rede limitando o transporte de valores superiores a um determinado limite mínimo por célula.
- valor de disparo (`accutriggerflux/state`) – permite o fluxo de material apenas se um valor de disparo for excedido.

2.2.2 - Modelo Integrado Multi-Escala / RIKS

Autômatos celulares são geralmente utilizados para modelar processos de mudança do uso e cobertura do solo. Tradicionalmente, autômatos celulares são implementados segundo critérios estritamente locais, isto é, a dinâmica de aplicação das regras de transição baseiam-se principalmente na vizinhança de uma célula. Entretanto, em muitos casos de processos urbanos, a função de transição deve levar em conta diferentes fatores, incluindo: os efeitos da vizinhança, a qualidade do solo (fator ambiental), as taxas demográficas da região (fator social), a demanda por uma determinada atividade econômica e o comportamento dos agentes econômicos.

Na literatura recente, verifica-se uma tendência de propostas de extensões ao modelo de autômato celular clássico, visando integrar fatores ambientais e sócio-econômicos, para representar a dinâmica espacial de fenômenos urbanos.

Entre estas propostas destaca-se a do *Research Institute for Knowledge Systems* (RIKS 2001), que apresenta uma estrutura de modelagem dinâmica e de suporte a decisão capaz de operar em uma variedade de escalas (Engelen, 1995).

A arquitetura do modelo Multi-Escala é constituída de dois sub-sistemas denominados macro e micro-escalas. Na macro-escala estão representadas as variáveis ecológicas e sócio-econômicas que afetam o sistema como um todo. A micro-escala representa a dimensão espacial do modelo. Estas escalas interagem intensamente entre si e com um Banco de Dados Geográfico, a partir do qual o modelo obtém os dados necessários para as simulações (Figura 2.9).

A macro-escala possui três componentes representando os subsistemas natural, econômico e social. Estes subsistemas estão conectados através de uma rede de influência mútua e recíproca. O subsistema natural representa condições ambientais tais como temperatura, precipitação e poluição. O subsistema social inclui dados demográficos como nascimentos, morte e migração. O subsistema econômico é fortemente determinado pelas mudanças ocorridas no subsistema natural e pelas demandas sociais. Neste sentido, ele pode gerar demandas como, por exemplo, a necessidade por mais células residenciais quando a população aumenta.

A micro-escala consiste em um autômato celular sobre o qual são aplicadas regras de transição para calcular as mudanças no uso do solo.

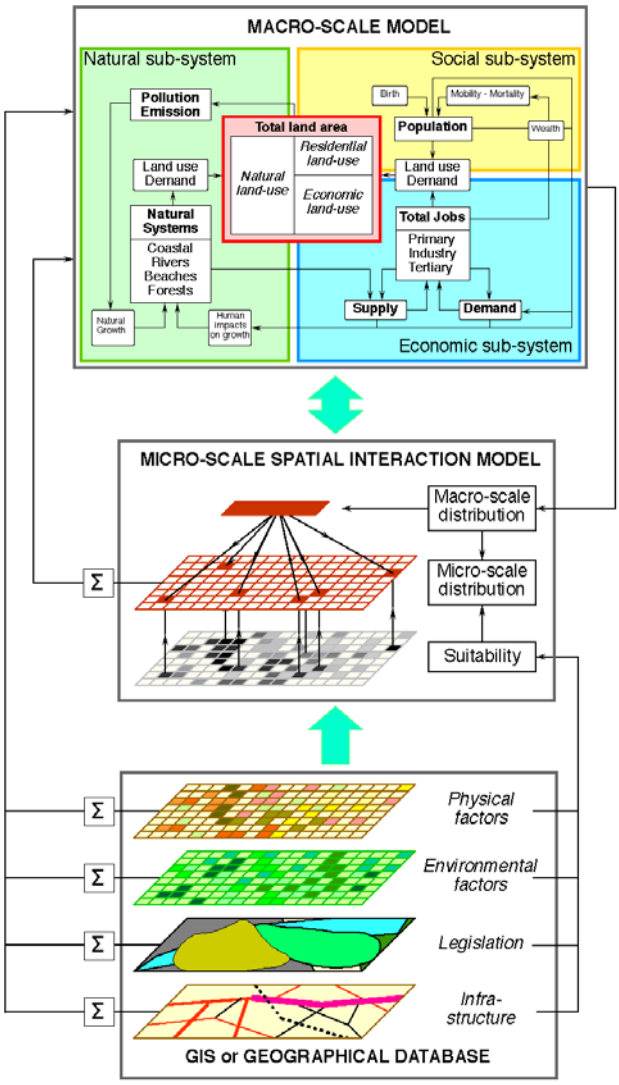


FIGURA 2.9 - Integração entre o Modelo Multi-Escala/RIKS e GIS.
FONTE: Engelen (1995).

Neste autômato, os estados das células representam categorias de uso do solo (Figura 2.10) e são divididos em duas categorias: funções e feições. Funções são usos do solo ativos, tais como residencial, floresta, comercial. Em princípio, uma célula função pode mudar para qualquer um dos estados possíveis. Feições são usos do solo fixos, tais como rios, parques e aeroportos. Embora as feições não estejam sujeitas às mudanças geradas pelas regras de transição do autômato celular, eventualmente elas podem ser convertidas através de um processo especial ou uma intervenção exógena. Feições aparecem como argumentos das regras e podem afetar

a transição de células vizinhas. Assim, por exemplo, a existência de um parque pode influenciar a transição de células vizinhas em células residenciais (White, 1997).

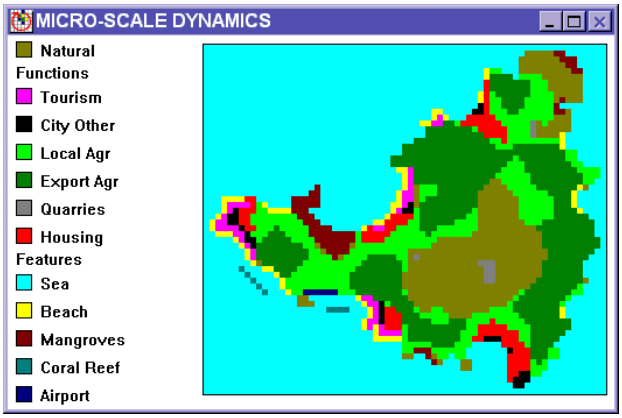


FIGURA 2.10 - A micro-escala.

A vizinhança de uma célula é uma região circular com um número variável de células, organizadas em zonas de distâncias. A Figura 2.11 representa uma célula com vizinhança circular de diâmetro 8, que abrange uma área de 197 células, organizadas em 30 zonas de distâncias. As zonas de distâncias influenciam o efeito atração/repulsão que as células vizinhas têm no potencial de transição de uma célula.

						30								
				28	27	25	24	25	27	28				
			29	26	23	21	20	19	20	21	23	26	29	
		29	25	22	18	16	15	14	15	16	18	22	25	29
		26	22	17	14	13	11	10	11	13	14	17	22	26
	28	23	18	14	12	9	8	7	8	9	12	14	18	23	28
	27	21	16	13	9	6	5	4	5	6	9	13	16	21	27
	25	20	15	11	8	5	3	2	3	5	8	11	15	20	25
30	24	19	14	10	7	4	2	1	2	4	7	10	14	19	24	30
	25	20	15	11	8	5	3	2	3	5	8	11	15	20	25	
	27	21	16	13	9	6	5	4	5	6	9	13	16	21	27	
	28	23	18	14	12	9	8	7	8	9	12	14	18	23	28	
		26	22	17	14	13	11	10	11	13	14	17	22	26		
		29	25	22	18	16	15	14	15	16	18	22	25	29		
			29	26	23	21	20	19	20	21	23	26	29			
					28	27	25	24	25	27	28					
								30								

FIGURA 2.11 - Vizinhança circular.

Na macro-escala estão as variáveis ambientais e sócio-econômicas do modelo. Neste nível, as funções buscam estabelecer relações de causa e efeito entre estas variáveis, a fim de gerar a demanda por serviços e produtos, que determina uma nova configuração para uso e cobertura do solo.

O subsistema natural representa condições ambientais tais como temperatura, precipitação e poluição. Neste subsistema, o usuário define as relações de influência

entre as variáveis. Funções que descrevem mudanças na temperatura e no nível do mar ao longo do tempo, por exemplo, são utilizadas para projetar demandas externas por produtos da área em estudo.

No subsistema social, dados demográficos como nascimentos, morte e migração são utilizados para modelar o crescimento populacional. O crescimento populacional, associado a informações sobre a qualidade de vida e condições de trabalho, provenientes do subsistema econômico, são parâmetros essenciais para calibrar o modelo, gerando a demanda por células residenciais e de determinadas atividades e serviços.

Esta demanda por células, gerada pela macro-escala, é aplicada à micro-escala (autômato celular) de acordo com um mecanismo baseado em três classes de prioridades (Engelen et al., 1997):

- **Regras de prioridade 1** são intervenções do usuário como, por exemplo, a inclusão de um aeroporto.
- **Regras de prioridade 2** são regidas pelo subsistema natural e geram certas transições diretamente, sem interferência do autômato celular. Por exemplo, se o nível do mar sobe, células com baixa elevação são convertidas em praias ou mangues (White e Engelen, 1997).
- **Regras de prioridade 3** se aplicam às células ativas (funções). Para cada célula ativa é calculado um vetor de potencialidades, em que cada potencialidade representa o grau de atração de uma célula para um determinado estado (z).

O potencial P_z de uma célula, para uma atividade z, é calculado através da seguinte expressão (White e Engelen, 1997) :

$$P_z = S_z . N_z + \varepsilon_z$$

onde:

- S_z expressa a adequabilidade da célula para a atividade Z $(0 \le S_z \le 1)$
- N_z expressa o efeito de vizinhança da célula para a atividade $Z (0 \le N_z \le 1)$
- ε_z é uma pertubação estocástica (Gaussiana)

O efeito agregado da vizinhança N para a atividade z é calculado por:

$$N_z = \sum_{d,i} I_{d,i} W_{z,y,d}$$

onde:

d = zona de distância

i = índice das células na zona de distância d

$w_{z,y,d}$ = parâmetro de peso aplicado a células no estado y na zona de distância d ($0 \leq d \leq 30$)

$I_{d,i}$ = 1, se a célula i na distância d está no estado y; 0, caso contrário.

A adequabilidade de uma célula é uma medida da capacidade desta célula em suportar a atividade z, calculada a partir de uma combinação linear de suas características ambientais e físicas, tais como: topografia, qualidade do solo, e precipitação. Os valores de adequabilidade são normalizados entre zero (totalmente inadequado) a um (adequado) e são constantes durante a execução do modelo. O modelo Multi-Escala/RIKS não implementa funções para calcular a adequabilidade. Assim, para gerar dados de adequabilidade que possam ser utilizados para alimentar o modelo, deve-se utilizar um GIS, com este tipo de funcionalidade.

Células ativas são convertidas para o estado para o qual seu potencial é maior, mas só até que a demanda por células deste estado seja atendida. Depois deste ponto, nenhuma outra célula é convertida para este estado. Para realizar este controle, o modelo Multi-Escala/RIKS utiliza em suas formulações matemáticas o conceito de densidade do solo, que é o número de pessoas que podem morar ou trabalhar em uma célula. Esta medida varia no tempo em função da demanda por uma atividade e a disponibilidade de solo (células) para esta atividade e está diretamente relacionada com o parâmetro de adequabilidade da célula (Engelen et al., 1993).

2.2.3 - Resumo dos ambientes computacionais

Processos físicos e de uso e cobertura do solo possuem mecanismos distintos para aplicação de regras de transição. Enquanto os processos físicos podem ser descritos por modelos determinísticos, os processos de uso e cobertura do solo são caracterizados como processos estocásticos e são altamente influenciados por variáveis exógenas.

Nas seções anteriores foram apresentadas as principais características de dois ambientes computacionais para modelagem dinâmica, PCRaster e Modelo Multi-

Escala/RIKS. Para finalizar este capítulo, apresentaremos a seguir um quadro comparativo dos principais aspectos computacionais destes sistemas:

TABELA 2.2 – Quadro comparativo PCRaster x Modelo Multi-Escala/RIKS.

GIS	PCRaster	Multi-Escala/RIKS
Estrutura de Dados	- mapas (raster) - LDD - séries temporais	- autômato celular - vizinhança circular - dados multi-escala
Estrutura de Controle	- Timer	- regras de transição - células feição/função
Funcionalidade	- modelo determinístico - Funções de transporte sobre redes hidrológicas (accuflux, accucapacity accuthreshold, accufraction) - Timeinput,lddcreate - cellarea, maptotal	- modelo estocástico - potencialidade de uma célula para uma atividade - acessibilidade à rede de transporte - demandas externa e interna para um produto/atividade
Interface	- interface de caracteres - ambiente não integrado - entrada de dados e acesso a funções via scripts	- interface Gráfica - ambiente integrado - interface gráfica para entrada e edição dos dados e funções

CAPÍTULO 3
AMBIENTE COMPUTACIONAL PARA MODELAGEM DINÂMICA ESPACIAL

TerraML é um acrônimo para TerraLib *Modeling Language*. TerraLib é uma biblioteca de código aberto, em desenvolvimento no Instituto Nacional de Pesquisas Espaciais (INPE). TerraLib fornece no seu núcleo (Figura 3.1) funcionalidade para manipular dados geográficos e facilidades para conversão de dados, visualização e gerenciamento de banco de dados espacial (Câmara et al., 2000). Algoritmos que usam as estruturas do núcleo incluindo análise espacial, linguagens de consulta e de simulação, e rotinas para conversão de dados, também estão disponíveis.

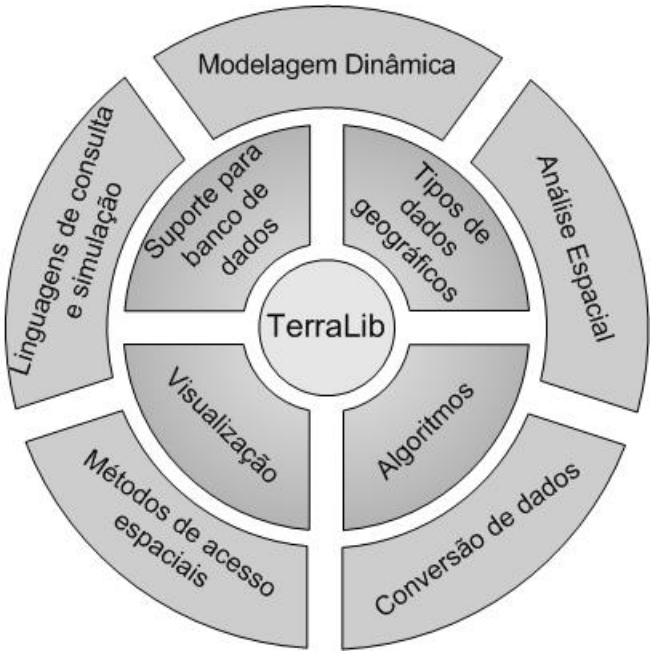


FIGURA 3.1 - A estrutura da TerraLib.

Neste capítulo será apresentado o ambiente computacional para modelagem ambiental desenvolvido nesta tese. O capítulo está organizado em duas partes: aspectos conceituais, que apresenta os fundamentos teóricos sobre os quais este ambiente foi projetado, e aspectos de implementação, que apresenta as estruturas de

dados e tecnologia utilizada para implementar o ambiente computacional para dar suporte à linguagem TerraML.

3.1 - Aspectos Conceituais

3.1.1 - Espaço Celular

Modelos celulares têm sido largamente utilizados em aplicações de GIS tais como, dinâmica do uso do solo, ocupação urbana do solo e processos físicos de fluxo e transporte de material, especialmente em hidrologia, entre outras aplicações. As motivações para o uso deste modelo de dados são influenciadas pela natureza "pixelizada" do dado remoto e pela conveniência da programação e implementação de estruturas baseadas em grades (O´Sullivan, 2000).

A computação celular se baseia em três princípios: simplicidade, paralelismo e localidade. A célula, unidade fundamental de um sistema celular, possui um conjunto de atributos extremamente simples e realiza poucas tarefas. Um aspecto poderoso desta estrutura de dados é o alto grau de conectividade existente entre as células. Nestes sistemas, uma célula pode se comunicar com suas vizinhas, trocando informações. Entretanto, nenhuma célula tem uma visão geral do sistema, o que caracteriza estes sistemas como descentralizado. Portanto, sistemas celulares têm como aplicações potenciais sistemas em que a ordem global emerge de ações locais, tais como sistemas biológicos e físicos. O paralelismo é um ponto controverso na computação celular, assim como em outras áreas da computação. O paralelismo é desejado, mas dificilmente atingido em um grau satisfatório, devido à falta de ambientes computacionais estritamente paralelos. Entretanto, mecanismos artificiais permitem acelerar o desempenho de tais sistemas através da sincronização das ações (Sipper, 1999).

Num espaço celular as relações espaciais entre as células se dão no espaço próximo, onde as relações de vizinhança desempenham um papel determinante. O tamanho e a configuração (forma) de uma vizinhança podem variar de sistema para sistema, de modelo para modelo, e de variável para variável. Entretanto, na maioria dos sistemas existentes a vizinhança é tratada de forma estacionária, i.e., os vizinhos de uma célula

estão sempre dispostos nas mesmas direções (norte, sul, leste, oeste). Outro ponto importante no modelo celular é a necessidade de gerenciar uma variedade de escalas. As operações celulares podem se dar em escala local (por célula), zonal (por região) e global (por plano de informação). A integração de processos multi-escala pode ser facilmente implementada com distribuição uniforme dos dados. Entretanto, processos mais complexos requerem mecanismos não uniformes de distribuição para balancear demandas e restrições (White, 1997).

Em TerraML o espaço celular é definido como uma estrutura de dados matricial, onde cada célula contém vários atributos. As células podem ser manipuladas como objetos geográficos individuais e operações projetadas para objetos podem ser aplicadas a elas.

3.1.2 - Modelo Temporal

Bancos de dados temporais consistem em aplicações de banco de dados que representam algum aspecto de tempo ao organizar suas informações. Num banco de dados temporal, o tempo é considerado como uma seqüência ordenada de pontos em alguma granularidade, determinada pela aplicação.

Em TerraML, o suporte temporal para banco de dados é implementado através do **versionamento de objetos.** Nesta abordagem, os atributos dinâmicos são associados a atributos temporais de **tempo inicial** e **tempo final**. Sempre que é atualizado um atributo dinâmico, em vez de sobrescrever valores, como ocorre em banco de dados não temporais, o sistema cria uma nova versão. Esta nova versão é adicionada como uma nova tupla no banco de dados (Elmasri e Navathe, 2002). Veja na Tabela 3.1 os diferentes usos do solo em 3 células no período de 01-01-1989 a 31-12-1991. A primeira célula (C000L000) tem dois usos do solo: uso 0, de 01-01-1989 a 31-12-1989; uso 1 de 01/01/1990 a 31/12/1991. A célula C000L001 tem apenas um uso do solo, uso 0, de 01/01/1989 a 31/12/1991. Já a célula C000L002 tem um uso do solo diferente para cada ano (1989, 1990, 1991) e, portanto, tem três versões de tuplas. Note que as mudanças no uso do solo são assíncronas, i.e., não ocorre em intervalos regulares. Num sistema de atualização de dados síncrono, todos os objetos teriam o mesmo número de versões.

TABELA 3.1 Tabela de um BD Temporal utilizando versionamento de objetos.

Object_id	Land_use	Initial-Time	Final_Time
C000L000	0	1989-01-01	1989-12-31
C000L000	1	1990-01-01	1991-12-31
C000L001	0	1989-01-01	1991-12-31
C000L002	1	1989-01-01	1989-12-31
C000L002	2	1990-01-01	1990-12-31
C000L002	0	1991-01-01	1991-12-31

Para manipular as múltiplas versões de um mesmo objeto, um sistema para banco de dados temporais deve prover operadores temporais que permitam obter a primeira, a corrente, a próxima e última versão de um objeto. Permitindo assim, percorrer o banco através de critérios de busca temporais como, por exemplo:

- Obtenha o uso do solo do objeto C000L000 **em 1989**?
- Obtenha o uso do solo **corrente** do objeto C000L000?

3.1.3 - Modelo de Mudança

Nos últimos anos, os conceitos de autômatos celulares têm sido utilizados para modelar fenômenos físicos e urbanos (Batty, 1999; Burrough, 1998; Roy, 1996; Engelen, 1995; Câmara, 1996). Um autômato celular consiste de uma grade uniforme e regular, onde cada célula pode assumir um estado discreto. Autômatos celulares evoluem em passos de tempo discretos, em que as células são atualizadas, simultaneamente, com base no estado das células em sua vizinhança, no passo de tempo anterior, e de acordo com um conjunto de regras locais. A vizinhança de uma célula é definida pela célula em si e todas as células imediatamente adjacentes. (Wolfram, 1983).

Para ilustrar como se dá o mecanismo de aplicação das regras de transição, apresentaremos um exemplo simples baseado em (Câmara, 1996). Neste exemplo, uma célula pode assumir dois estados (branco e preto) e sua vizinhança é definida sobre duas células adjacentes. As regras de transição especificam que o estado de uma célula num instante t+1 é igual ao dos seus vizinhos no instante t, se estes

vizinhos tiverem os estados iguais; caso contrário, o estado da célula permanece o mesmo. Para entender o exemplo é necessário identificar os componentes básicos do autômato celular clássico, que são (Engelen et al., 1997):

- espaço euclidiano, dividido em uma grade de células
- uma vizinhança de tamanho e formato definidos (Figura 3.2a)
- um conjunto de estados discretos (Figura 3.2b)
- um conjunto de regras de transição (Figura 3.2c)
- um conjunto de intervalos de tempo, com atualização simultânea das células (Figura 3.2d)

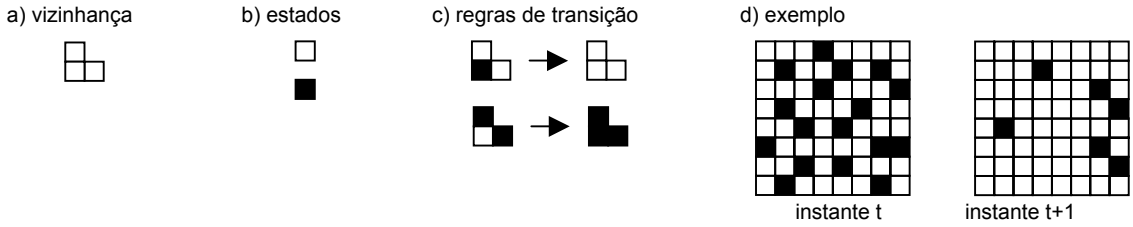


FIGURA 3.2 - Exemplo de autômato celular.
FONTE: Câmara (1966).

A dinâmica de aplicação das regras de transição em um autômato celular é semelhante à de um filtro espacial. Desta forma, todas as células são avaliadas e, quando for o caso, modificadas para um novo estado. Na figura 3.2 d, a primeira célula da segunda linha do autômato tem, no instante t, o estado branco e suas vizinhas possuem estados diferentes (uma é branca e outra preta). Neste caso o estado da célula permanece o mesmo (1ª regra de transição). Seguindo o mesmo mecanismo, a segunda célula da segunda linha tem, no instante t, o estado preto e suas vizinhas têm, ambas, o estado branco, logo o estado desta célula sofre uma transição para branco (2ª regra de transição). O processo segue este mecanismo para as demais células até que todas tenham sido avaliadas.

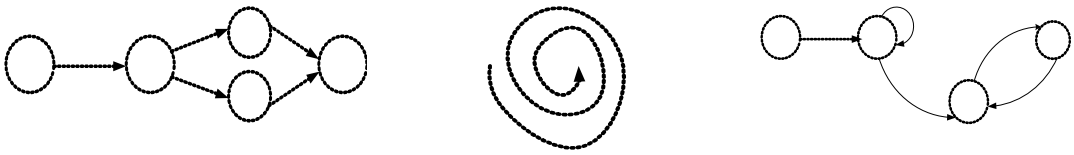
No exemplo acima, pode-se observar que as mudanças geradas por autômatos celulares são estritamente locais, isto é, baseadas nas vizinhanças de cada célula. Nesta perspectiva, pode-se dizer que sua aplicação é eficiente em processos em que a ordem global emerge de ações locais e descentralizadas. Entretanto, em Sistemas de Informações Geográficas, a ordem global depende tanto de fatores endógenos (ações locais) como exógenos (ações à distância). Uma abordagem para modelar

tais sistemas é o conceito de espaço celular (*cell space*), uma variação do autômato celular clássico (*strict cellular automata*), que flexibiliza a questão da regularidade do espaço e localidade das vizinhanças (Batty, 2000).

Autômatos celulares, na sua forma tradicional ou em versões estendidas, têm sido utilizados para implementar modelagem dinâmica ambiental na maioria dos sistemas disponíveis atualmente (White e Engelen, 1997; Sipper, 1999; Soares Filho e Cerqueira, 2002). Entretanto, a limitação deste modelo computacional tem sido reconhecida em várias publicações (Couclelis, 1997; O'Sullivan, 2002), que identificam os seguintes fatores como os pontos limitantes deste modelo:

- A estacionariedade da sua vizinhança. Nestes modelos todas as células têm a mesma configuração de vizinhança. Assume-se a isotropia espacial.
- O espaço de estados é finito. Apenas mudanças discretas, determinadas por atributos igualmente discretos podem ser representadas.
- Ações locais. As transições de estados das células ocorrem fundamentalmente como conseqüência do estado das suas células vizinhas.

Um dos maiores desafios para o desenvolvimento de uma linguagem para suportar modelagem ambiental é a necessidade de representar um sistema de transição com componentes discretos e contínuos. Para este propósito, o paradigma de autômato celular clássico não é suficiente. Entretanto, existe uma solução baseada em autômatos chamada de autômatos híbridos. Um autômato híbrido é um modelo formal para representar sistemas dinâmicos com componentes discretos e contínuos, Figura 3.3, que são atualizados em uma seqüência de passos (Henzinger, 1996).



a) Discreto b) Contínuo c) Híbrido

FIGURA 3.3 - Dinâmica de sistemas.

Autômatos celulares híbridos são autômatos finitos com variáveis reais, que são atualizadas através de uma dinâmica contínua, dentro de um espaço de estados

infinito. Um autômato híbrido H consiste dos seguintes componentes (Henzinger, 1996):

- **Variáveis:** um conjunto finito $X = \{x_1,..., x_n\}$ de variáveis reais, onde n é a dimensão do H. Escrevemos *X* para o conjunto $\{x_1,..., x_n\}$ de variáveis reais (que representam derivadas de primeira ordem durante mudanças contínuas), e escrevemos *X'* para o conjunto $\{ x'_1, ..., x'_n\}$ de variáveis iniciais(que representam valores na conclusão da mudança discreta)
- **Grafo de Controle:** um multi-grafo direcionado finito (*V,E*). Os vértices em *V* são chamados modos de controle e as arestas *E* são chamadas de interruptores de controles (*control switches*).
- **Condição Inicial:** um predicado cujas variáveis livres pertencem a *X*.
- **Condições Invariantes**: um predicado cujas variáveis livres pertencem a *X*.
- **Condições de fluxo:** um predicado cujas variáveis livres pertencem a $X \cup X'$
- **Condições de mudança:** uma aresta rotulada que atribui a cada interruptor de controle $e \in E$ um predicado. Cada condição de mudança é um predicado cujas variáveis livres pertencem a $X \cup X'$
- **Eventos:** um conjunto finito Σ de eventos; e uma aresta rotulada $E \rightarrow \Sigma$ que atribui a cada interruptor de controle um evento.

Na Figura 3.4 estão representados alguns dos componentes do autômato Híbrido. A condição inicial representa uma condição para entrar em um dos modos de controle de controle do autômato, no início do sistema. Condições invariantes são condições que mantém um modo de controle ativo. As Condições de fluxo expressam os predicados que são executados em cada modo de controle. Condições de mudança são utilizadas para representar mudanças discretas entre os modos de controle. Cada condição de mudança é ligada a uma aresta do grafo.

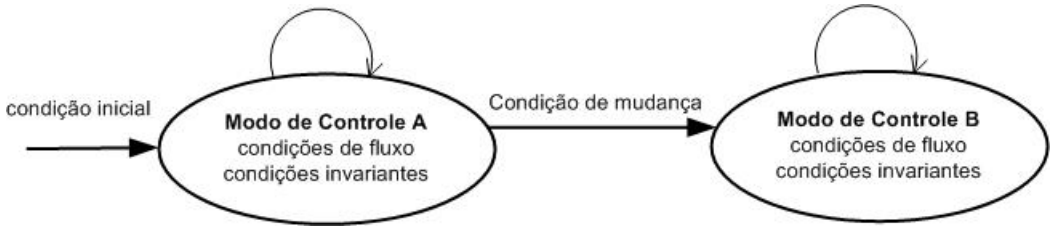


FIGURA 3.4 – Esquema geral de um autômato híbrido.

Para explicar o conceito de autômato híbrido no domínio das aplicações geográficas vamos considerar um caso de balanço hídrico do solo. Neste sistema, séries temporais de precipitação são utilizadas para preencher as células com água da chuva

até que a capacidade de infiltração da célula seja atingida, quando então ocorre o fluxo da água excedente. Na Figura 3.5, um grafo de controle ilustra o mecanismo pelo qual o autômato de balanço hídrico evolui.

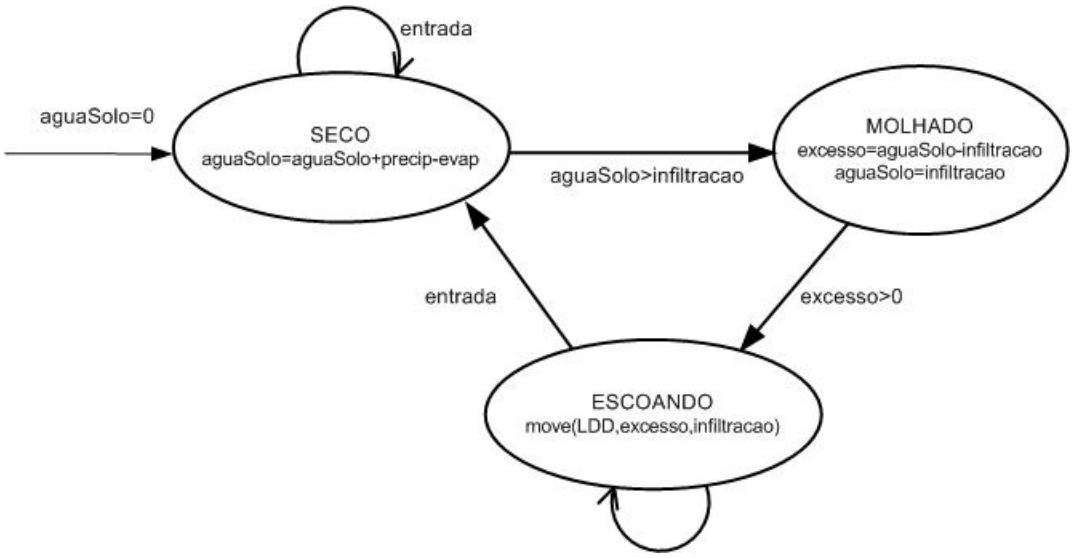


FIGURA 3.5 – Autômato balanço hídrico.

O autômato de balanço hídrico tem uma condição inicial (`aguaSolo=0`). Quando uma série temporal é lida, através do evento `entrada`, o valor da água no solo é calculado. Em seguida, é verificado se a água no solo atingiu a capacidade de infiltração da célula. Se a água no solo for maior que a capacidade de infiltração da célula, a água excedente é calculada e transportada para outra célula. Esta trajetória é repetida em todos os passos da simulação. Os principais elementos do autômato de balanço hídrico estão resumidos na Tabela 3.2.

TABELA 3.2 Resumo do autômato balanço hídrico.

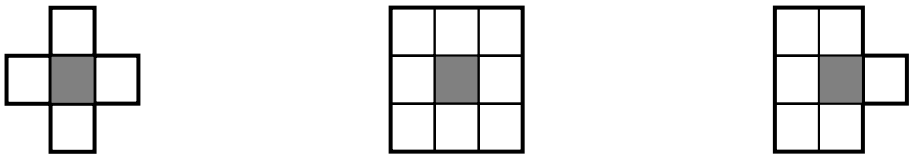
Modos de Controle	Condições de Fluxo	Condições de Mudança	eventos	Transição para
SECO	aguaSolo(t+1)=aguaSolo(t)+precip+evap	aguaSolo>infiltracao		MOLHADO
MOLHADO	excesso=aguasolo-infiltracao	excesso>0		ESCOANDO
ESCOANDO	MOVE(LDD,excesso, infiltracao)	excesso>0	entrada	SECO

Na teoria de autômatos híbridos, eventos são ações dependentes do contexto da aplicação. No caso deste autômato de balanço hídrico, em particular, o evento `entrada` representa a ação de ler uma série temporal de precipitação (`precip`). É a

partir deste evento que o valor de água no solo (`aguaSolo`), de uma célula, pode ser atualizado. `Move` é uma função que transporta a água excedente (`excesso`) através de uma rede de drenagem (`LDD`), tendo como parâmetro a capacidade de infiltração das células (`infiltração`).

3.1.4 - Vizinhança Generalizada

Em autômatos celulares clássicos, as transições de estado são realizadas tendo como base o estado das células em uma vizinhança pré-determinada. Em tais sistemas, prevalece a estacionariedade das vizinhanças, i.e., todas as células tem a mesma configuração de vizinhança, e as ações são essencialmente locais. Exemplos clássicos de vizinhanças são: Moore, von Neumann e assimétrica, representadas abaixo na Figura 3.6.



a) von Neumann b) Moore c) Assimétrica

FIGURA 3.6 - Exemplos de vizinhança.

Entretanto, em muitos fenômenos do mundo real, relações de influência se estabelecem com base em redes de transporte. Como exemplo, considere o processo de mudança no uso do solo da Amazônia. Este processo é condicionado pela ocupação urbana, que por sua vez, é fortemente determinada pela rede de transporte (rios e rodovias) da região, como mostra a Figura 3.7 (rede de transporte esta marcada em vermelha e os assentamentos urbanos em branco). Neste caso, um modelo realista para mudança no uso do solo tem que considerar definições flexíveis de proximidade para capturar ações à distância.

Em um modelo celular, uma matriz de proximidade para representar as relações de vizinhança entre as células mostra-se uma solução apropriada para dar ao modelo a flexibilidade necessária para capturar ações à distância.

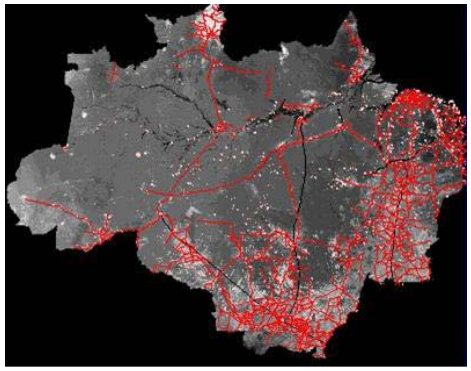


FIGURA 3.7 - Ocupação urbana e rede de transporte da Amazônia.

A matriz de vizinhança generalizada é uma variação da matriz de proximidade (*spatial weights matrix*) onde os pesos são calculados a partir de relações espaciais no espaço absoluto, tais como distância euclidiana e adjacência, ou com base em relações espaciais no espaço relativo, que levam em conta a conectividade de objetos em uma rede de transporte ou de comunicação, por exemplo. Nesta visão, a combinação dos conceitos de espaço absoluto e relativo conduz à implementação do conceito de espaço próximo (Couclelis, 1997), onde a informação espacial é geo-referenciada no espaço absoluto, mas está também associada a uma representação do espaço relativo do qual é parte.

Uma Matriz de Vizinhança Generalizada (MVG) é composta de um conjunto de objetos espaciais O, um grafo G e uma matriz de proximidade V.

Objetos espaciais representados por células, de formatos e tamanhos regulares, dispostas em uma grade, no caso do ambiente computacional TerraML.

Grafo conjunto nós e arcos, em que cada nó representa um objeto (célula, no caso do autômato) e os arcos representam relacionamentos de vizinhança entre dois nós.

Matriz de proximidade composta de um conjunto de elementos W_{ij} que servem para indicar o quanto dois objetos O_i e O_j estão próximos. Geralmente é representada em termos de adjacência ou distância euclidiana. As opções mais comuns para definir W_{ij} são:

a) $W_{ij} = 1$, se O_i é vizinho de O_j; caso contrário $W_{ij} = 0$.
b) $W_{ij} = 1$, se a distância(O_i, O_j) < maximum; caso contrário $W_{ij} = 0$.
c) $W_{ij} = 1/\text{distance}\ (O_i, O_j)^2$; se i=j, $W_{ij} = 0$.

3.2 - Aspectos de Implementação

Em termos de implementação, o espaço celular da TerraML pode ser dividido em duas partes chamadas (1) estrutura básica e (2) estrutura estendida (Figura 3.8). A estrutura básica é estática e representa o conjunto de atributos que toda célula tem independente do modelo . A estrutura estendida é dinâmica, i.e., definida em tempo de execução para acomodar os dados fornecidos por um documento TerraML.

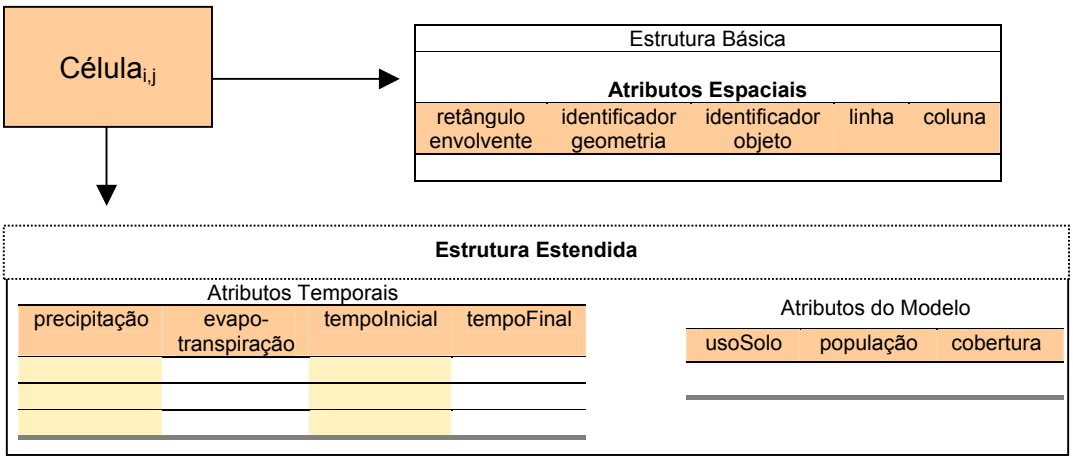


FIGURA 3.8 - A estrutura de dados celular.

A estrutura básica é essencialmente espacial. Cada célula tem seu `retângulo envolvente`, seu endereço no espaço celular (`linha, coluna`), e outros atributos tais como os identificadores de objeto (`identificador objeto`) e de geometria (`identificador geometria`).

A estrutura estendida contém atributos que variam de acordo com a aplicação. Por esta razão, estes atributos são criados e ligados à estrutura da célula via um mecanismo de alocação dinâmica de memória. Estes atributos referem-se às características físicas, ambientais e sócio-econômicas das células, podendo ser temporais. Atributos temporais são aqueles que têm múltipla ocorrência na célula tais como os diferentes usos do solo ao longo do tempo, por exemplo. Eles são implementados com um suporte de banco de dados temporal para gerenciar suas múltiplas versões. Na Figura 3.8, os atributos temporais, `precipitação` e `evapo-transpiração`, e os atributos do modelo, `usoSolo`, `população` e `cobertura`, são apenas ilustrativos. Como estes atributos são definidos em tempo de execução, não

há como deixar definidos *a priori* os seus nomes. Estes atributos só são conhecidos quando o banco de dados é lido, durante a execução do processador TerraML, que será descrito a seguir.

Um documento TerraML é compilado para um programa com chamadas de funções TerraLib. Para isto foi desenvolvido um compilador chamado Processador TerraML. O Processador TerraML lê um documento TerraML, analisa a correção sintática deste programa e gera um conjunto de funções a partir da Terralib (Figura 3.9).

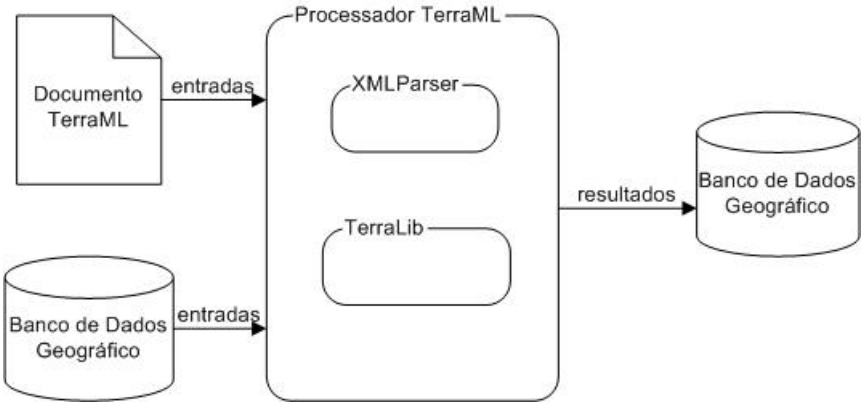


FIGURA 3.9 - Diagrama do TerraML Processor.

TerraLib está sendo implementada em linguagem C++ com uma abordagem orientada a objetos. Desta forma, a estrutura celular é implementada como uma hierarquia de classes, onde cada classe representa os principais componentes do espaço celular.

Na Figura 3.10 está representada a hierarquia de classes da representação da noção de espaço celular, seguindo a notação UML (Booch et al., 1998). Nesta figura, a classe principal é o espaço celular (`TeCellAutomata`) que é composto por uma grade celular (`BiCellSet`), que é um conjunto de células. A vizinhança de uma célula refere-se ao conjunto de células que influenciam o estado de uma célula. Este conjunto de células pode ter qualquer configuração, ser contíguo ou não, bem com ter qualquer número de células. Por isto, em TerraML, a vizinhança é implementada como uma matriz de proximidade (`TeProxMatrix`).

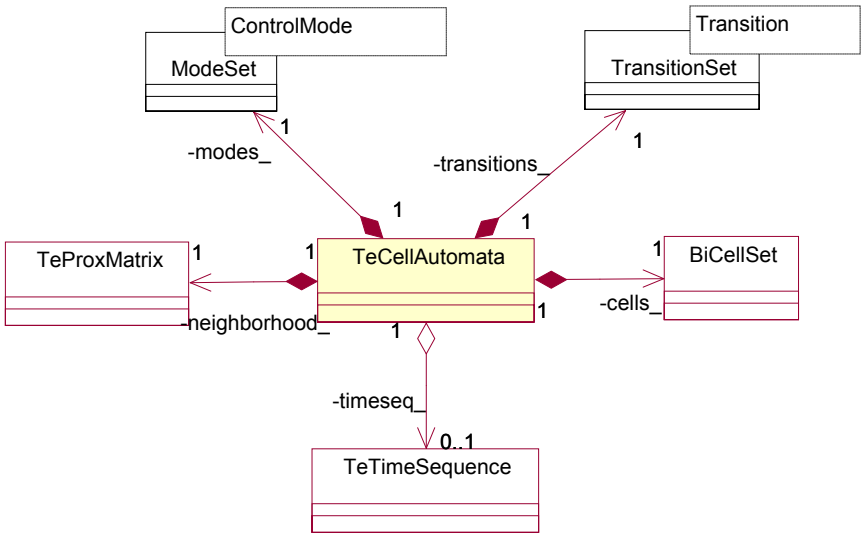


FIGURA 3.10 - Diagrama de classes do autômato híbrido (`TeCellAutomata`).

Em TerraML, usamos a abstração de um autômato híbrido para modelar as transições e modos de controle do sistema. Desta forma, além da vizinhança e de conjunto de células, um autômato híbrido possui um conjunto de modos de controle (`ModeSet`) e um conjunto de transições (`TransitionSet`).

Uma transição consiste na mudança de um modo de controle para outro, portanto, possui um modo de controle origem (`fromode_`) e destino (`toMode_`) (Figura 3.11). Cada transição é definida a partir de um conjunto de condições (`Conditions`) que têm que ser satisfeitas. Na abstração do autômato híbrido, estas condições são chamadas de condições de mudança. Os modos de controle (`ControlMode`) são constituídos de um conjunto de condições de fluxo. As condições de fluxo são equações matemáticas necessárias para representar o modelo, tais como média ponderada, lógica *fuzzy* e equações diferenciais, entre outras. Neste componente, TerraML estende a teoria de autômatos híbridos, não se restringindo às equações diferenciais expressar condições de fluxo.

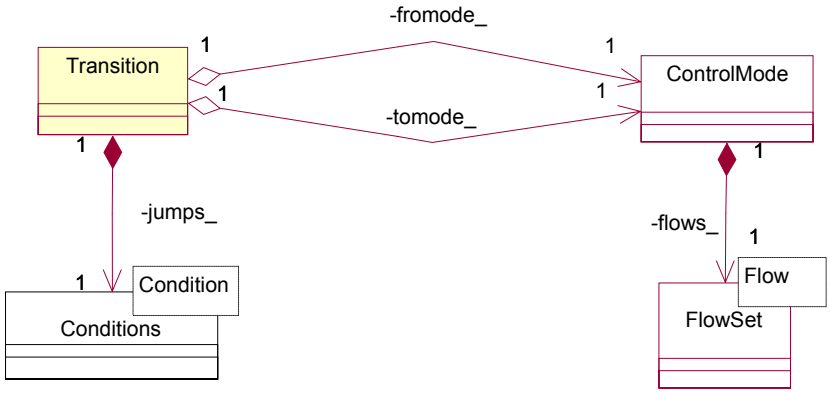


FIGURA 3.11 – Diagrama da classe `Transition`.

Como as equações das condições de fluxo variam bastante nos seus parâmetros (operandos de uma equação), a classe `Flow` é uma classe abstrata que pode assumir a forma de qualquer uma das suas classes derivadas.

Toda condição de fluxo `(flow)` tem um atributo (`attribute`), que corresponde ao nome do atributo da célula que receberá o resultado da equação. Este atributo é geralmente chamado de lado esquerdo da equação (*left hand side*).

Na Figura 3.12, está representada a implementação da célula do TerraML. Como já mencionado, a célula tem uma estrutura básica que corresponde aos seus atributos espaciais. Estes atributos incluem a são a identificação do objeto (`object_Id`) e Identificação da Geometria (`geomId_`), e suas coordenadas X (`column_`) e Y (`line_`) no espaço celular. Estas características das células são herdadas da classe `TeCell` e `TeGeometry`, implementadas na biblioteca TerraLib. A parte estendida da célula, inclui seus atributos dinâmicos (`dynAttribute_`) e atributos do modelo (`attribute_`), que variam de simulação para simulação, e são definidos na classe BiCell. Os atributos dinâmicos têm múltiplas versões, uma para cada intervalo da simulação.

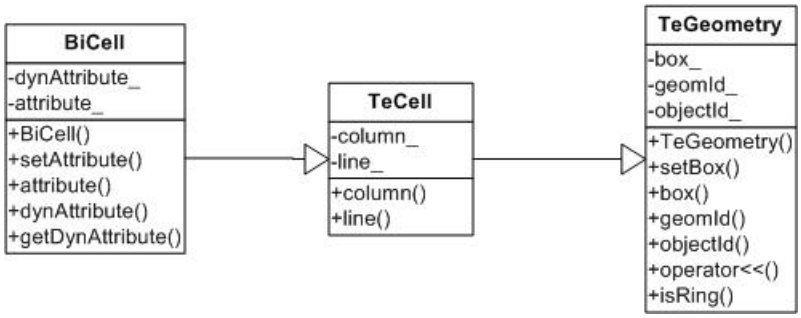


FIGURA 3.12- Diagrama da classe `Bicell`.

A vizinhança do autômato é implementada como uma matriz de proximidade, que por sua vez é implementada em uma estrutura de dados de matriz esparsa. Esta abordagem de implementação confere flexibilidade ao modelo, de tal forma que uma célula pode ter qualquer número de vizinhos. Cada célula da matriz de proximidade possui, além das identificações das células vizinhas (`object_id`), informações sobre a zona de distância (`slice_`) , peso de influência (`weight_`), distância dos centróides das células (`d1_`), distância das células à rede de transporte (`d2_`) e distância mínima entre as células vizinhas pela rede de transporte (`d3_`), como mostra a Figura 3.13. Os atributos `d2_` e `d3_` são utilizados apenas em casos de vizinhanças geradas a partir de redes de transportes (seção 3.1.4).

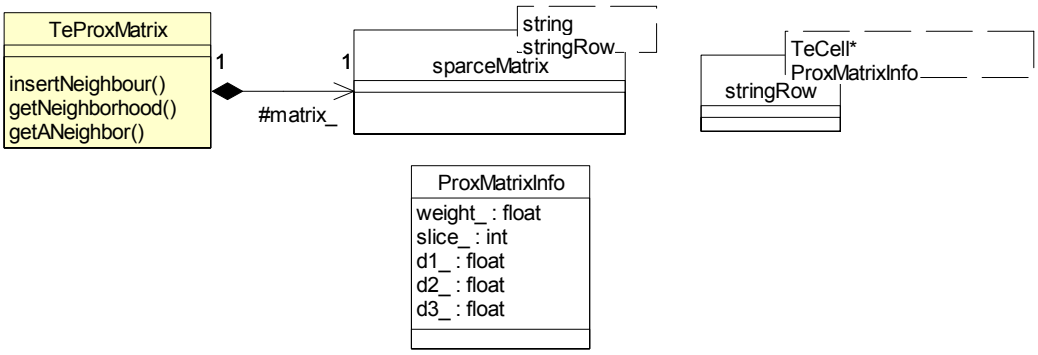


FIGURA 3.13 - Diagrama de classes da matriz de proximidade (`TeProxMatrix`).

O diagrama de classes da Figura 3.14 apresenta as estruturas de dados temporais do TerraML. Estas classes foram implementadas segundo os conceitos de granularidade

(TeChronon), intervalo (TeTimeSpan) e Seqüência (TeTimeSequence) que, no TerraML, serve para gerenciar as múltiplas versões de dados geradas durante a simulação, bem como controlar as estruturas de repetição das condições de fluxo do sistema.

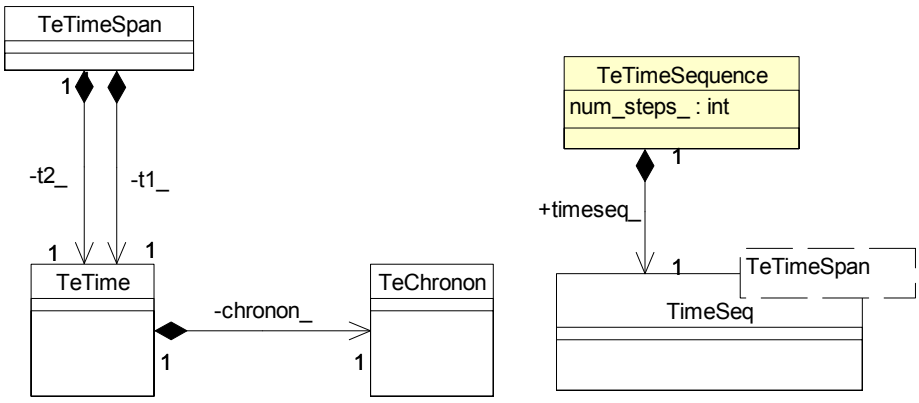


FIGURA 3.14 - Diagrama de classes da seqüência de tempo (TeTimeSequence).

CAPÍTULO 4
TERRAML - FORMALIZAÇÃO DA LINGUAGEM

TerraML é uma linguagem para descrever modelos dinâmicos espaciais, que tem como arcabouço os três conceitos chaves descritos no capítulo anterior: a) a teoria de autômatos híbridos, (b) a vizinhança generalizada e (c) o modelo celular.
A seguir serão descritos os principais elementos desta linguagem, através das suas especificações em DTD, e de suas funcionalidades, demonstradas através de algoritmos.

4.1 - Arquitetura

TerraML é uma linguagem baseada em XML, que é um acrônimo para *e**X**tensible **M**arkup **L**anguage. Markup Languages* são linguagens para marcação de textos. A sintaxe destas linguagens é caracterizada pela utilização de *tags* (palavras encapsuladas por sinais '<' e '>'). XML é uma meta-linguagem, i.e., uma linguagem de sintaxe aberta que pode ser utilizada para definir outras linguagens. As vantagens da criação de novas linguagens a partir da XML são:
1) obter independência de plataforma, e
2) usufruir da disponibilidade de uma família de ferramentas, que oferecem serviços úteis para a realização de tarefas de desenvolvimento importantes tais como *parsers* e ambientes de desenvolvimento (IDE).
TerraML é organizada hierarquicamente em seções. Um programa escrito em TerraML é composto de uma seção principal chamada *Cell Processor,* que é subdividida em 2 sub-seções: *input e control* (Figura 4.1). Cada sub-seção da linguagem TerraML inclui um conjunto de elementos com significado especial para o processador. A seção *input* especifica o conjunto dos dados a serem recuperados a partir de um banco de dados e atribuídos às células como atributos. Funciona como um *binding,* um programa para ler dados e ligá-los às variáveis do programa. Na seção *control* são definidos os modos de controle do autômato e as regras de transições para o sistema alternar entre estes modos de controle.

Neste capítulo todas as seções e elementos da linguagem TerraML serão descritos através de um diagrama (quando o elemento for composto por outros elementos), um DTD, um dicionário de dados e um exemplo de uso. DTDs, definições do tipo documento, são a forma básica e mais amplamente utilizada para especificações de tipo para documentos XML. Através de um DTD pode-se especificar os tipos de elementos que podem aparecer em um documento, a sua ordem e atributos, e seu conteúdo.

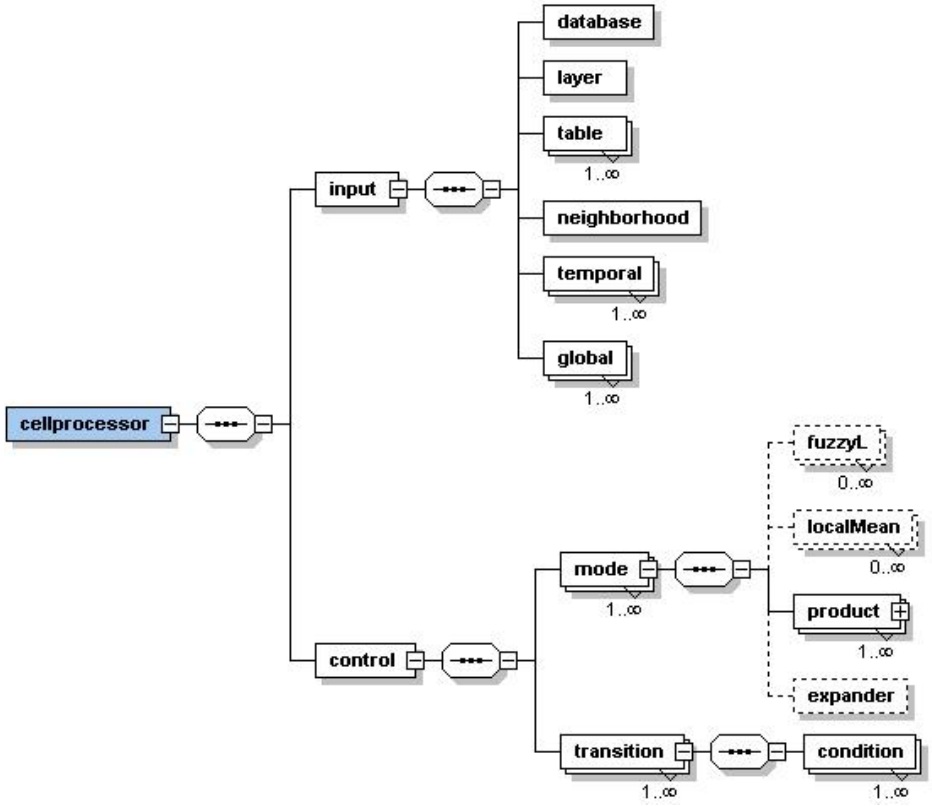
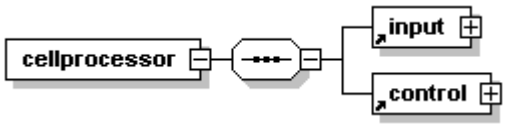


FIGURA 4.1 - Arquitetura da linguagem TerraML (notação W3C Schema/XMLSPY).

Na Tabela 4.1 é especificada a seção `CellProcessor`. Esta seção é composta de duas sub-seções `input` e `control`. A seção `CellProcessor` tem alguns atributos para identificação do usuário e da aplicação para a qual o programa Terra. Estes atributos devem ser preenchidos com o propósito de documentar o documento.

TABELA 4.1 Especificação da seção `CellProcessor`.

<table>
<tr><td>Diagrama</td><td> 
</td></tr>
<tr><td>Dicionário de Dados</td><td>
<table>
<tr><th>Nome</th><th>Tipo</th><th>Uso</th><th>Descrição</th></tr>
<tr><td>author</td><td>string</td><td>obrigatório</td><td>Autor do programa TerraML</td></tr>
<tr><td>date</td><td>string</td><td>obrigatório</td><td>Data do programa</td></tr>
<tr><td>model</td><td>string</td><td>obrigatório</td><td>Modelo da Simulação</td></tr>
</table>
</td></tr>
<tr><td>DTD</td><td>
<pre>
<!ELEMENT cellprocessor (input, control)>
<!ATTLIST cellprocessor
    author   CDATA #IMPLIED
    date   CDATA #IMPLIED
    model   CDATA #IMPLIED
>
</pre>
</td></tr>
</table>

4.2 - Especificação da Linguagem

4.2.1 - A Seção de Entrada de Dados

Em TerraML as variáveis que serão lidas, a partir de um dispositivo de armazenamento, e associadas às células são definidas na seção `input` e podem ser: mapas ou imagens, séries temporais, tabelas, variáveis escalares e matrizes de proximidade.

A Tabela 4.2 apresenta o diagrama (notação W3C do software XMLSPY) e o DTD da seção `input`.

Banco de Dados

Em TerraML, os atributos das células são preenchidos a partir de um banco de dados. Como conseqüência, na seção `input`, o primeiro elemento a ser especificado deve ser o banco de dados. Para isto, devem ser fornecidos os atributos apresentados na Tabela 4.3.

TABELA 4.2 – Especificação da seção `Input`.

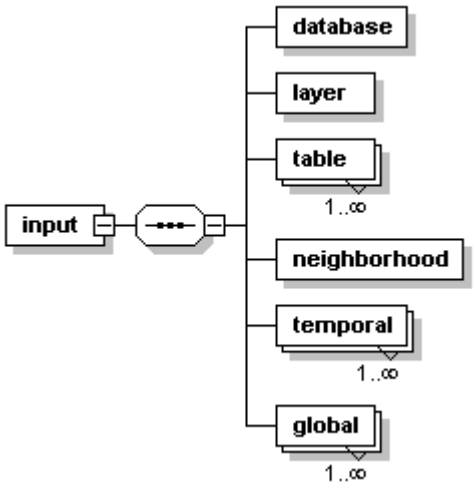
Diagrama	
DTD	<!ELEMENT input (database, layer, table, neighborhood, global+)>
Dicionário de Dados	Veja a definição dos elementos da seção *input* nas tabelas: database Tabela 4.3 layer Tabela 4.4 table Tabela 4.5 neighborhood Tabela 4.6 temporal Tabela 4.7 global Tabela 4.8

TABELA 4.3 - Especificação do elemento `database`.

DTD	<!ELEMENT database EMPTY> <!ATTLIST database host CDATA #REQUIRED path CDATA #REQUIRED name CDATA #REQUIRED user CDATA #IMPLIED pass CDATA #IMPLIED >
Dicionário de Dados	**Nome** / **Tipo** / **Uso** / **Descrição** host / string / obrigatório / nome do servidor de arquivos path / string / obrigatório / caminho de navegação pela estrutura de arquivos name / string / obrigatório / nome completo (nome.extensão) do arquivo user / string / opcional / Identificação do usuário pass / string / opcional / Senha do usuário
Exemplo	< **database host**="localhost" **path**="c:/tese/" **name**="rondonia.mdb" **user**="" **pass**=" />

Layer

No modelo de dados TerraLib os atributos geométricos das células estão armazenados no banco de dados como um `layer`, uma tabela com atributos geográficos. Para que TerraML possa recuperar os atributos espaciais das células, têm que ser especificados o nome (`name`) e identificação (`layerid`) desta tabela, seguindo a sintaxe apresentada na Tabela 4.4, a seguir:

TABELA 4.4 Especificação do elemento `layer`.

DTD	<!ELEMENT layer EMPTY> <!ATTLIST layer name CDATA #REQUIRED layerid CDATA #REQUIRED >
Dicionário de dados	**Nome** **Tipo** **Uso** **Descrição** name string obrigatório nome da tabela layerid inteiro obrigatório identificação da tabela
Exemplo	**<layer name**="celulas450" **layerid**="46" />

Tabelas

No modelo de dados TerraLib os atributos não-espaciais das células estão armazenados no Banco de dados em uma tabela, que tem que vir especificada na seção `input`, para que TerraML possa recuperar estes atributos.

TABELA 4.5 Especificação do elemento `table`.

DTD	<!ELEMENT table EMPTY> <!ATTLIST table name CDATA #REQUIRED columns CDATA #REQUIRED lines CDATA #REQUIRED >
Dicionário de dados	**Nome** **Tipo** **Uso** **Descrição** name string obrigatório nome da tabela columns inteiro obrigatório número de colunas do espaço celular lines ineiro obrigatório número de linhas do espaço celular
Exemplo	**< table name**="celulas450" **columns**="35" **lines**="70" />

Séries Temporais

Séries temporais são dados para os quais podem existir vários valores em diferentes instantes de tempo. Estes dados são associados às células como atributos temporais, aqueles que tem múltiplas ocorrências e, como conseqüência, requerem mecanismos especiais para controle de versões. Para que as séries temporais possam ser recuperadas e associadas às células, devem ser fornecidas as seguintes informações:

TABELA 4.6 – Especificação do elemento `temporal`.

DTD	<!ELEMENT temporal EMPTY> <!ATTLIST temporal name CDATA #REQUIRED attribute CDATA #REQUIRED period CDATA #REQUIRED timeunit (YEAR | MONTH | DAY | HOUR) #REQUIRED >
Dicionário de dados	**Nome** **Tipo** **Uso** **Descrição** name string obrigatório nome completo (nome.extensão) do arquivo attribute string obrigatório nome da coluna period inteiro obrigatório número de períodos da série (ex: 12 meses) timeunit string obrigatório granularidade temporal (mês, ano, dia)
Exemplo	<**temporal** **name**=”precip99” **attribute**=”precip” **period**=”3” **time**=”month” />

No exemplo da tabela acima é especificado que a tabela `precip99` contém dados pluviométricos de `3 meses`. O atributo temporal da célula se chama `precip`.

Variáveis Globais

Variáveis escalares representam valores numéricos ou alfanuméricos. A principal característica de uma variável escalar em TerraML é a sua generalidade. Ela tem o mesmo valor em qualquer ponto do espaço, isto é, o mesmo valor para qualquer célula. Devido a esta propriedade, a declaração de uma variável escalar é precedida pelo *tag* <global>. No exemplo da Tabela 4.7 é definida a variável `global` altitude. Isto significa que todas as células têm altitude 500.

TABELA 4.7 – Especificação do elemento `global`.

DTD	<!ELEMENT global EMPTY> <!ATTLIST global name CDATA #REQUIRED value CDATA #REQUIRED >
Dicionário de dados	**Nome** / **Tipo** / **Uso** / **Descrição** name / string / obrigatório / nome pelo qual a variável será referenciada no programa value / real / obrigatório / constante numérica (real)
Exemplo	**<global name**=”altitude” **value**=”500” />

Vizinhança

Em TerraML os dados de vizinhança de uma célula correspondem a uma matriz generalizada de proximidade. Esta matriz tem que ser gerada externamente e importada pelo ambiente TerraML. Para realizar a importação dos dados de vizinhança e ligá-los às células na forma de atributo, o dado de vizinhança deve ser informado de acordo com a especificação da Tabela 4.8.

TABELA 4.8 Especificação do elemento `neighborhood`.

DTD	<!ELEMENT neighborhood EMPTY> <!ATTLIST neighborhood name CDATA #REQUIRED zones CDATA #REQUIRED >
Dicionário de dados	**Nome** / **Tipo** / **Uso** / **Descrição** name / string / obrigatório / nome do arquivo de vizinhança (ascii) zones / int / obrigatório / número de zonas de distância
Exemplo	**<neighborhood name**="c:/tese/vizinho3.txt" **zones**="3" />

No exemplo da tabela acima é especificado que a vizinhança das células deve ser lida a partir do arquivo `c:/tese/vizinho3.txt`, e que devem ser consideradas apenas `3 zonas` de distâncias para os vizinhos de uma célula.

4.2.2 - A seção de controle

A seção de controle da TerraML encapsula as dinâmicas discreta e contínua de um autômato híbrido. Desta forma, os principais elementos desta seção são os modos de controle e as transições. A Figura 4.2 apresenta o diagrama W3C *schema* da seção `control`.

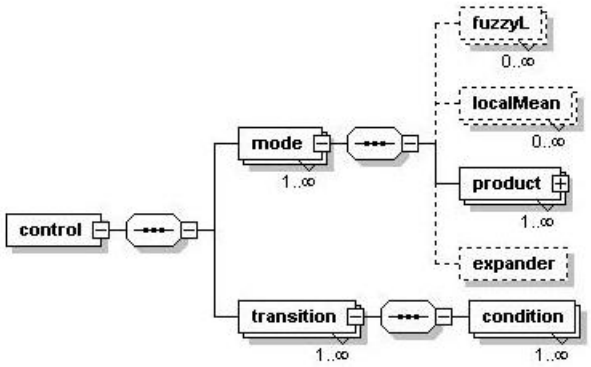


FIGURA 4.2 - Esquema geral da seção *Control.*

Parâmetros de tempo

A seção `control` tem início com a declaração das variáveis temporais do modelo, que estão detalhadas na tabela 4.9. Estes parâmetros são fundamentais para controlar o início, final e passo de tempo das simulações.

No exemplo da tabela acima está sendo especificado que a simulação tem início no dia (`timeunit`) de 01 de maio de 2003 (`initime`), e será realizada por 30 (`intervals`) dias, variando dia a dia (`step`).

Condições de fluxo

Os modos de controle encapsulam as condições de fluxo, que são equações executadas para todas as células em cada passo da simulação. Em TerraML, cada modo de controle (`mode`) tem um nome (`name`), e um conjunto de condições de fluxo conforme mostrado na Tabela 4.10. As condições de fluxo podem assumir diferentes formatos e ter diferentes formas de implementação (algoritmos). TerraML oferece um conjunto de funções muito utilizadas em modelagem dinâmica espacial, tais como média local, função *fuzzy* e média ponderada, entre outras.

TABELA 4.9 – Especificação da seção `control`.

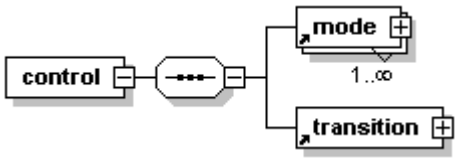
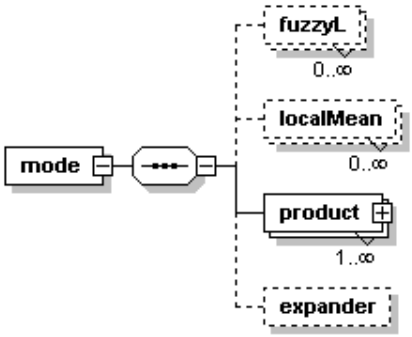
Diagrama	
Dicionário de dados	**Nome** / **Tipo** / **Uso** / **Descrição** initime / string / obrigatório / tempo inicial da simulação intervals / inteiro / obrigatório / número de intervalos de tempo da simulação step / inteiro / obrigatório / intervalo de tempo entre cada simulção timeUnit / string / obrigatório / granularidade temporal (ano , mês, dia, hora)
DTD	<!ELEMENT control (mode)> <!ATTLIST control initime CDATA #REQUIRED intervals CDATA #REQUIRED step CDATA #REQUIRED timeUnit (YEAR | MONTH | DAY | HOUR) #REQUIRED >
Exemplo	<control **initime**="05/01/03" **intervals**="30" **step**="1" **timeUnit**="DAY" />

TABELA 4.10 - Especificação do elemento `mode`.

Diagrama	
Dicionário de dados	**Nome** / **Tipo** / **Uso** / **Descrição** name / string / obrigatório / nome do modo de controle fuzzyL / fuzzyL / opcional / ver Tabela 4.12 localMean / localMean / opcional / ver Tabela 4.11
DTD	<!ELEMENT mode (fuzzyL+, localMean+, product+, expander+)> <!ATTLIST mode name CDATA #REQUIRED >
Exemplo	< **mode name**=”Global”

Média Local

A equação média local é utilizada para calcular a influência dos vizinhos de uma célula. Sua equação e algoritmo de implementação estão apresentados na seção 4.3. Na Tabela 4.11, a seguir, estão especificada a sintaxe desta equação em TerraML.

TABELA 4.11 – Especificação do elemento `localMean`.

DTD	<!ELEMENT localMean EMPTY> <!ATTLIST localMean attribute CDATA #REQUIRED column CDATA #REQUIRED >
Dicionário de dados	**Nome** / **Tipo** / **Uso** / **Descrição** attribute / string / obrigatório / atributo da célula que receberá o resultado da expressão column / string / obrigatório / atributo da célula sobre o qual será calculada a média
Exemplo	**<localMean attribute** = "atratividade" **column** = "cobertura_solo" />

Neste exemplo é especificado que a `atratividade` de uma célula é calculada através de uma média local sobre o atributo `cobertura_solo` das células vizinhas. O endereço das células vizinhas e o peso de influência de cada uma delas estão na matriz de proximidade especificada na seção `input`.

Funções *fuzzy*

Funções *fuzzy* são utilizadas para caracterização de classes que não apresentam limites bem definidos entre si (Burrough e McDonnel, 1998). Funções fuzzy podem ser expressas de várias formas. A função `FuzzyL` implementada em TerraML tem a seguinte formulação:

$$f(x) = \frac{1}{\left(1 + \alpha(x - \beta)^2\right)} \quad (0 \le x \le P) \text{ e } (0 \le f(x) \le 1)$$

onde:

- x é um atributo da célula
- α é um parâmetro que governa a forma da função (senoidal, linear)
- β é um parâmetro que indica o valor do ponto central da função

A Tabela 4.12 apresenta a sintaxe para expressar uma equação *fuzzy* em TerraML.

TABELA 4.12 – Especificação do elemento `fuzzyL`.

DTD

```
<!ELEMENT fuzzyL EMPTY>
<!ATTLIST fuzzyL
   attribute   CDATA   #REQUIRED
   column      CDATA   #REQUIRED
   alpha       CDATA   #REQUIRED
   beta        CDATA   #REQUIRED
>
```

Dicionário de dados

Nome	Tipo	Uso	Descrição
attribute	string	obrigatório	atributo da célula que receberá o resultado da expressão
column	string	obrigatório	atributo da célula sobre o qual será processada a função *fuzzy*
alpha	real	obrigatório	parâmetro que governa a forma da função
beta	real	obrigatório	valor central do resultado da função

Exemplo

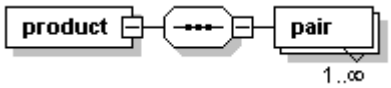
```
<fuzzyL attribute="acessibilidade"
      column="distancia_rodovia"
      alpha="0.001"
      beta="500"
/>
```

Produto

A equação produto é utilizada para calcular a média ponderada de um conjunto de atributos das células. A Tabela 4.13 apresenta desta equação.

TABELA 4.13 – Especificação do elemento `product`.

Diagrama



DTD

```
<!ELEMENT product (pair+)>
<!ATTLIST product
    attribute CDATA #REQUIRED
>
```

Dicionário de dados

Nome	Tipo	Uso	Descrição
attribute	string	obrigatório	nome do atributo que recebe o resultado
pair	pair	obrigatório	ver Tabela 4.14

Exemplo

```
<product attribute="potencial">
                <pair attribute="acessibilidade" weight="0.4"/>
                <pair attribute="atratividade"   weight="0.2"/>
</product>
```

Um par (`pair`) é uma combinação de um atributo da célula e seu respectivo peso, na média ponderada. Num produto pode aparecer qualquer número de pares. A sintaxe

do elemento pair é especificada na Tabela 4.14. O exemplo fornecido na tabela abaixo corresponde à expressão: $potencial = (acessibilidade * 0.4) + (atratividade * 0.2)$.

TABELA 4.14 – Especificação do elemento `pair`.

DTD	<!ELEMENT pair EMPTY> <!ATTLIST pair attribute CDATA #REQUIRED weight CDATA #REQUIRED >
Dicionário de dados	**Nome** / **Tipo** / **Uso** / **Descrição** attribute / string / obrigatório / atributo da célula sobre o qual será calculada a média ponderada weight / real / obrigatório / parâmetro de peso
Exemplo	<**product attribute**="potencial"> <**pair attribute**="acessibilidade" **weight**="0.4"/> <**pair attribute**="atratividade" **weight**="0.2"/> </**product**>

Expander

Expander é uma função para alocar ou modificar o atributo de uma célula segundo um determinado parâmetro e uma demanda definida. A Tabela 4.15 apresenta a sintaxe do *expander*. Nesta tabela é mostrado um exemplo em que o `uso do solo` de uma célula poder ser modificado de acordo com o seu `potencial`, respeitada a `demanda` (expressa em número de células) por mudanças. O algoritmo de implementação do *expander* é apresentado na seção 4.3.

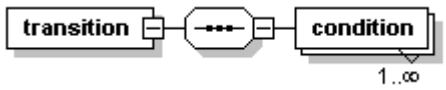
TABELA 4.15 – Especificação do elemento `expander`.

DTD	<!ELEMENT expander EMPTY> <!ATTLIST expander attribute CDATA #REQUIRED column CDATA #REQUIRED demand CDATA #REQUIRED >
Dicionário de dados	**Nome** / **Tipo** / **Uso** / **Descrição** attribute / string / obrigatório / atributo da célula que será alocado/modificado column / string / obrigatório / atributo da célula usado como parâmetro para a alocação/modificação demand / string / obrigatório / nome da variável global que expressa o número de células que podem ser alocadas/modificadas
Exemplo	<**expander attribute**="uso_solo" **column**="potencial" **demand**="demanda" />

Transições

As transições são mudanças de um modo de controle para outro, que são disparadas a partir da ocorrência de um evento. Portanto, cada transição é constituída de um modo de controle destino e um modo de controle de origem. Uma transição pode ser disparada a partir de uma ou mais condições.

TABELA 4.16 – Especificação do elemento `transition`.

<table>
<tr><td>Diagrama</td><td> 
</td></tr>
<tr><td>DTD</td><td><!ELEMENT transition (condition+ , event)>
<!ATTLIST transition
from CDATA #REQUIRED
to CDATA #REQUIRED
></td></tr>
<tr><td>Dicionário de dados</td><td><table>
<tr><th>Nome</th><th>Tipo</th><th>Uso</th><th>Descrição</th></tr>
<tr><td>from</td><td>string</td><td>obrigatório</td><td>modo de controle origem</td></tr>
<tr><td>to</td><td>string</td><td>obrigatório</td><td>modo de controle destino</td></tr>
<tr><td>condition</td><td>condition</td><td>opcional</td><td>ver Tabela 4.17</td></tr>
</table></td></tr>
<tr><td>Exemplo</td><td><transition from="GLOBAL" to="LOCAL">
<condition attribute="demanda" op="GT" value="0" />
</transition></td></tr>
</table>

No exemplo da Tabela 4.16 ocorre uma transição do modo de controle “GLOBAL” para “LOCAL” se a `demanda` for maior (`GT`) que `zero`.

Condição

As condições expressam o critério para que o autômato passe do modo de controle “from” para o modo de controle “to”. Em TerraML, condições são expressões relacionais do tipo apresentado na Tabela 17.

TABELA 4.17 - Especificação do elemento `condition`.

DTD

```
<!ELEMENT condition EMPTY>
<!ATTLIST condition
    attribute   CDATA   #REQUIRED
    op          CDATA   #REQUIRED
    value       CDATA   #REQUIRED
>
```

dicionário de dados

Nome	Tipo	Uso	Descrição
attribute	string	obrigatório	nome do attributo da célula sobre o qual será avaliada a condição
op	string	obrigatório	operador relacional (ver Tabela 4.18)
value	string	obrigatório	critério de comparação

Exemplo

```
<transition from="GLOBAL" to="LOCAL">
   <condition attribute="demanda"  op="GT"  value="0"   />
</transition>
```

No exemplo da Tabela 4.17 é especificada a condição de que a `demanda` seja `maior do que zero` para que ocorra a transição do modo de controle “GLOBAL” para “LOCAL”.

TABELA 4.18 – Operadores relacionais em TerraML.

Operador	Significado
GE	maior igual
GT	maior que
EQ	igual
LE	menor igual
LT	menor que
NE	diferente

4.3 - Funcionalidades

Nesta seção são apresentados algoritmos que implementam algumas das funções do TerraML, cujas sintaxes foram descritas na seção anterior.

4.3.1 - Média Local

A função **média local** calcula a atratividade da célula para uma determinado “estado” com base nos seus vizinhos. Os parâmetros desta função são: 1) o atributo da célula que receberá o resultado deste cálculo, e 2) o atributo sobre o qual será calculada a

influência dos vizinhos. Por exemplo: para calcular a atratividade da célula para desmatamento (um valor de uso do solo) devem ser passados os parâmetros “atratividade” (atributo da célula que receberá o resultado do `localmean`) e “uso do solo”, atributo que terá seu valor avaliado para as células vizinhas.

FIGURA 4.3 - Algoritmo da função LocalMean.

```
TeCellAutomata::localMean(Flow  f,int t)
{
  string  atributo1=f.attribute();
  string  atributo2=f.column();
  float   resultado=0.00;
  Para cada celula cel do automato
 {    Para cada celula vizinha de cel
      {  peso = Matriz_Proximidade (cel,vizinha).peso();
         resultado =peso * vizinha.atributo(atributo2,t-1);
         vizinha++;
      }
      cel.attributo(atributo1,t)=resultado;
      cel++;
   }
}
```

No algoritmo acima, para cada célula `cel` do autômato, obtém-se as suas células vizinhas. Para cada célula `vizinha` obtém-se na matriz de proximidade o peso de influência desta célula `vizinha` sobre `cel`. O resultado é calculado multiplicando-se o peso pelo valor do `atributo2` (uso do solo,por exemplo) da célula vizinha no instante de tempo `t-1`. O `atributo1` (atratividade, por exemplo) da célula, no instante de tempo `t`, é a somatória dos resultados (`peso` * `atributo2`) de todos os seus vizinhos no instante de tempo `t-1`.

4.3.2 - Expander

A função **expander** calcula o potencial global, que é a somatória de todas as células que tem potencial maior do que zero. Se o potencial global for maior que a demanda para aquele instante de tempo `t`, é calculado o limiar (`threshold`), que corresponde ao valor mínimo de potencial que uma célula tem que ter para ser modificada. Para calcular o limiar (`threshold`), as células são ordenadas (`sort`) por potencial. Desta ordenação, são selecionadas para transição as células de potencial mais alto. O número de células selecionadas é igual à demanda.

```
TeCellAutomata::expander(string attribute, double demand, int t)
{  int globalPotential =0;
   Para cada celula cel do automato
      Se cell.dynAttribute(attribute))>0)
         globalPotential++;
      cel++;

   double threshold = 0.00;
   Se (globalPotential>demand)
      threshold = sort(attribute, demand, t);
   transit(attribute, threshold, t);
}
```

FIGURA 4.4 - Algoritmo da função Expander.

4.3.3 - Funções *Fuzzy*

Como já mencionada na seção anterior, Tabela 4.12, uma função *fuzzy* em TerraML requer a definição de 4 parâmetros: `alpha`, `beta`, atributo (`attribute`) que recebe o resultado do processamento da função, e o atributo (`column`) sobre o qual a função é processada. O algoritmo da função FuzzyL, demonstrado na Figura 4.5, consiste em extrair da condição de fluxo (`f`) os parâmetros `attribute`, `alpha` e `beta`. Depois, para cada célula `cel`, calcular a expressão *fuzzy* e guardar o resultado deste cálculo `value` no atributo `column` da célula.

```
void TeCellAutomata::fuzzyL(fuzzyL f,int t)
{
      int attcol  =cells_.attribute(f.column());
      float alpha =f.alpha();
      float beta  =f.beta();
      double value;
      Para cada celula cel
      {
         value = (1 / (1 + alpha * pow (cell.attribute(attCol) - beta,2)));
         cel.dynAtt(t, cells_.dynAttribute(f.attr()), value);
         cel++;
      }
}
```

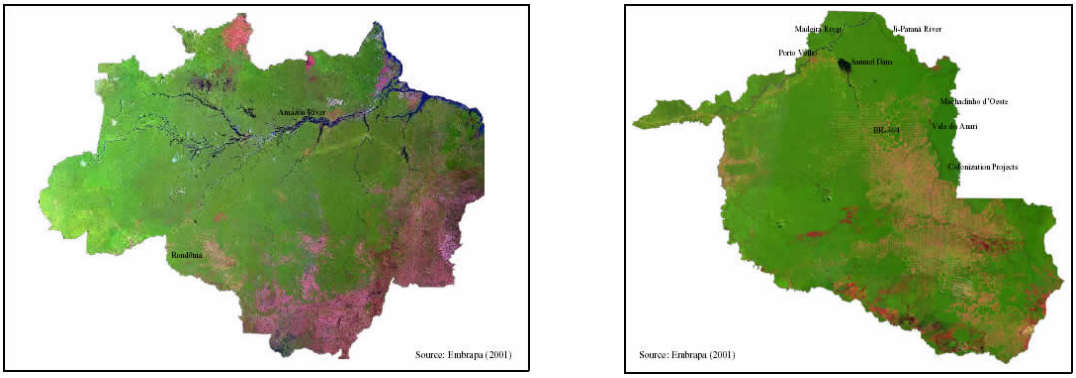
FIGURA 4.5 - Algoritmo da função FuzzyL.

CAPÍTULO 5
UMA APLICAÇÃO EM MUDANÇA NA COBERTURA DO SOLO

Neste capítulo, apresentaremos uma visão geral da TerraML através de um exemplo de um processo de mudança no uso e cobertura do solo relativo à evolução do desmatamento na Amazônia.

5.1 - Apresentação do Problema

O processo de desflorestamento e colonização da Amazônia Brasileira (Figura 5.1 a) tem sido associado às mudanças climáticas globais, à alteração dos ciclos biogeoquímicos, à dinâmica de uso e cobertura da terra e à diminuição da biodiversidade. Esta diversidade de fatores produz padrões heterogêneos de ocupação, o que torna complexa a modelagem de uma área extensa da floresta. Por esta razão, neste exemplo estaremos nos restringindo à uma pequena área do estado de Rondônia (Figura 5.1 b).



a) b)

FIGURA 5.1 A Amazônia (a) e o Estado de Rondônia (b).

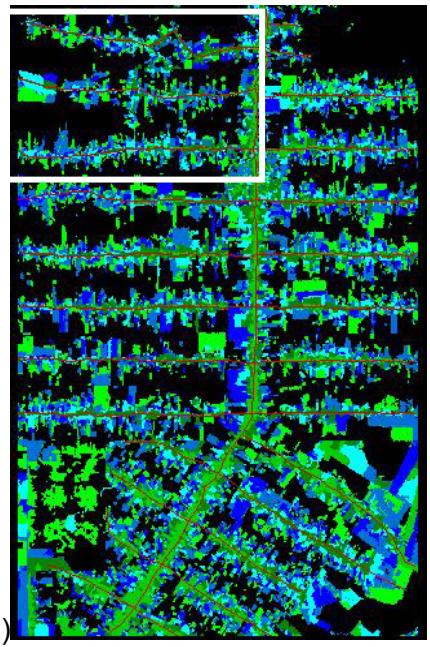
Em Rondônia cerca de 90% do desflorestamento tem-se concentrado nos limites de 100 km da malha rodoviária principal, ao redor dos eixos e pólos de desenvolvimento dos anos 70 e 80 (Alves, 2002). Assim, as áreas mais desmatadas ocorrem ao redor das estradas principais e secundárias provocando o fenômeno conhecido como “espinha-de-peixe”, que pode ser observado na figura 5.2. Nesta figura, as áreas em verde correspondem a floresta e as áreas em rosa correspondem a desmatamento.

As estradas são representadas por linhas vermelhas. Note que as áreas desmatadas se desenvolvem em sentido ortogonal à estrada principal.

FIGURA 5.2 Espinha-de-peixe.

Nesta aplicação consideraremos a área em destaque na Figura 5.3 b, que corresponde à uma área de 496.125 km, mapeados em 2.450 células, distribuídas em 70 colunas por 35 linhas. Cada célula representa uma área de 450m x 450m.

a)



b)

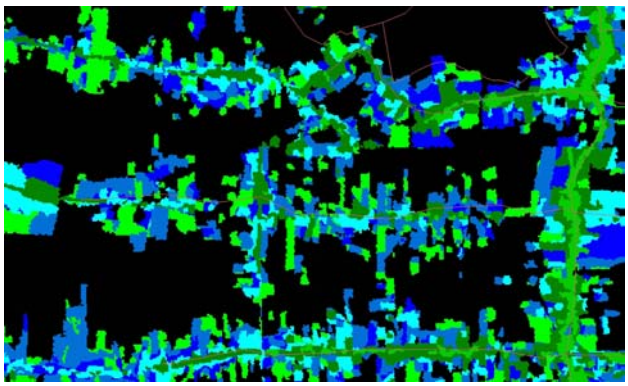


FIGURA 5.3 a) Detalhe da espinha-de-peixe b)zoom da área deste trabalho.

Neste exemplo, dois tipos de cobertura do solo são possíveis :

- Floresta (preto)
- Desmatado (colorido, cada cor representa o desmatamento em um ano)

5.2 - Metodologia

Na Figura 5.4, estão representadas as várias etapas seguidas desde a concepção do modelo até a sua execução. Algumas dessas etapas são repetidas ao longo das simulações e podem gerar dados que servem para realimentar o sistema.

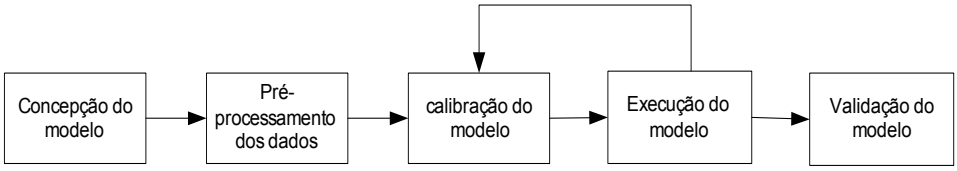


FIGURA 5.4 – Metodologia para simular mudanças em Rondônia.

Na **concepção do modelo,** são definidas as equações que determinam o comportamento do modelo. Nesta aplicação, o estado de uma célula pode mudar em função da seguinte combinação de fatores uma demanda global fornecida, propriedades da célula (acessibilidade), e a influência de sua vizinhança (atratividade). Em seguida, ocorre o **pré-processamento dos dados**, dados são recuperados e convertidos para um formato adequado para o processamento do modelo.

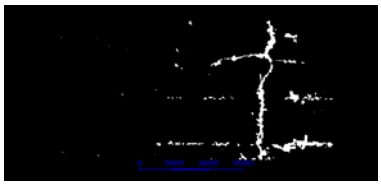
Durante a **calibração do modelo** são calculadas as variáveis utilizadas como parâmetros para a realização de ajustes. Essas variáveis expressam a demanda por ou restrições a determinadas mudanças. O Potencial de uma célula para mudança é um atributo dinâmico, i.e., que muda ao longo das várias simulações, por isto, tem que ser recalculado a cada iteração. Além disto, a demanda tem que ser atualizada sempre que ocorrem transições. A etapa **execução do modelo** consiste no processamento dos dados em si.

A **validação do modelo** é uma etapa complementar, consiste na comparação dos resultados gerados com dados reais.

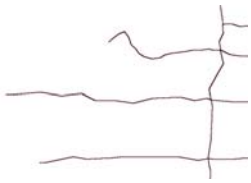
5.3 - Entradas

Para este modelo foram utilizadas as seguintes informações como entradas do sistema:

- Imagem do uso do solo em 1985 (Figura 5.5 a)
- Mapa da rede de transporte 1985 e 1988 (rodovias principais e secundárias) (Figura 5.5 b)
- Mapa das unidades de conservação em 1985 (Figura 5.5 c)
- Vizinhança (matriz generalizada de vizinhança)



a) Cobertura do solo 1985



B) Estradas

c) Unidades de Conservação

FIGURA 5.5 – Os atributos das células.

A imagem do uso do solo é utilizada para gerar as células e, juntamente com as demais entradas, preencher os atributos das células (Figura 5.6).

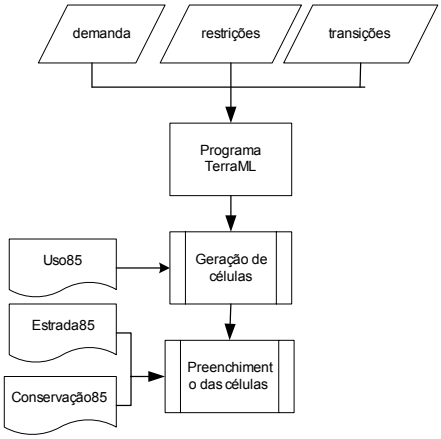


FIGURA 5.6 – Entradas do sistema.

A vizinhança da célula em TerraML é definida a partir de uma matriz de proximidade, como discutido no capítulo 4. Desta forma, é possível capturar o fenômeno espinha-de-peixe pois pode-se definir como vizinhos de uma célula outras células ligadas à

rede de transporte. Na Figura 5.7 é apresentado um esquema simplificado das vizinhanças de uma célula deste exemplo. Zonas de distâncias, representadas em cores diferentes, mostram os diferentes pesos de influência de cada célula vizinha.

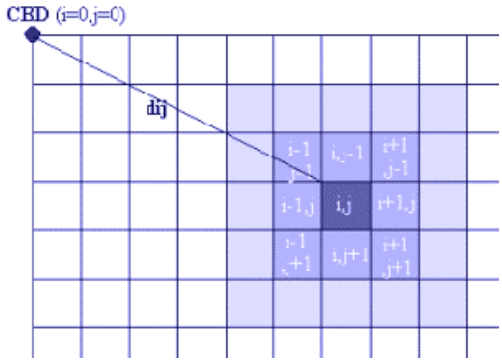


FIGURA 5.7 – Matriz de proximidade baseada em distância.

5.4 - Modelo

Nesta aplicação do TerraML definimos um modelo matemático baseado no modelo multi-escala desenvolvido pelo RIKS e apresentado no capítulo 2, seção 2.2.2. Neste modelo, o potencial de uma célula para o desflorestamento é calculado através da seguinte expressão:

$$\text{Potencial} = \text{Acessibilidade} \,.\, \text{Atratividade}$$

A acessibilidade de uma célula é uma medida da proximidade de uma célula em relação a uma rede de transporte e é expressa pela função *fuzzy*:

$$Acessibilidade = \frac{1}{\left(1 + \alpha(d - \beta)^2\right)} \qquad para\ 0 \le d \le P$$

onde:

- d é a distância euclidiana da célula à rodovia mais próxima
- α é um parâmetro que governa a forma da função (senoidal, nesta aplicação)
- β é um parâmetro que indica o valor do ponto central da $Acessibilidade$

A atratividade de uma célula para o desmatamento é calculada por:

$$Atratividade = \sum_{k} peso * uso \qquad 0 \leq Atratividade \leq 1$$

onde:

peso = parâmetro de peso aplicado a célula vizinha não desmatada
k = número de vizinhos da célula
uso = 1, se a célula vizinha está desmatada; 0, caso contrário.

5.5 - Restrições

As restrições entram no modelo como um instrumento de controle. Nesta aplicação temos como restrição a demanda pelo desmatamento. Deixado a si mesmo, o sistema poderia expandir o desmatamento a níveis exagerados, o que não corresponderia à realidade. A demanda por células desmatadas é um valor pré-determinado, uma entrada que deve ser informada no documento TerraML. Esta informação é útil para controlar o número de transições por passos. A variável dinâmica atratividade é um mecanismo auxiliar de controle, pois serve de parâmetro para selecionar as células a serem alteradas. Neste exemplo foi utilizada uma imagem obtida no ano 2000 para determinar a demanda.

5.6 - Especificação em TerraML

A Figura 5.8 apresenta o código TerraML para esta simulação. A explicação detalhada de cada uma das seções deste código é apresentadas nas próximas seções.

5.6.1 - Especificação dos dados de entrada

Na seção *input* estão relacionados os principais dados a serem recuperados do banco de dados e o arquivo de vizinhança.

```
<cellprocessor author="bianca" date="11/06/03" model="LUCC Rondonia" >
   <input>
      <database host="localhost" path="c:/tese_dados/"
                name="rondonia.mdb" user="" pass=""/>
      <layer name="celulas450" layerid="46"/>
      <table name="celulas450_dinamica" columns="35" lines="70"/>
```

```
      <neighborhood name="c:/tese_dados/vizinho1.txt" zones="1"/>
      <global name="demanda_total" value="700"/>
      <global name="demanda" value="0"/>
   </input>
   <control initime="1985" intervals="16" step="1" timeUnit="year">
      <mode name="global">
         <product attribute="demanda">
            <pair attribute="demanda" weight="0.0625"/>
         </product>
      </mode>
      <mode name="local">
         <fuzzyL attribute="acessibilidade1"   column="distancia_rod_princ"
                 alpha="0.001" beta="500"/>
         <fuzzyL attribute="acessibilidade2"   column="distancia_rod_secund"
                 alpha="0.001" beta="500"/>
         <localMean attribute="atratividade" column="cobertura_solo"/>
         <product attribute="potential">
            <pair attribute="acessibilidade1"  weight="0.4"/>
            <pair attribute="acessibilidade2"  weight="0.4"/>
            <pair attribute="atratividade"   weight="0.2"/>
         </product>
         <expander attribute="cobertura_solo" column="potencial" demand="demanda"/>
      </mode>
      <transition from="global" to="local">
         <condition attribute="demanda" op="GT" value="0"/>
      </transition>
   </control>
</cellprocessor>
```

FIGURA 5.8 – Um exemplo em TerraML para mudança no uso e cobertura do solo.

Neste exemplo, o banco de dados onde estão armazenadas as células é o arquivo `rondonia.mdb`, um banco Access. Neste banco a tabela `celulas450` é o layer a partir do qual pode ser obtida a geometria das células (Tabela 5.1), e a tabela `celulas450_dinamica` é a tabela a partir da qual podem ser obtidos os atributos das células, tais como cobertura do solo (`cobertura_solo`), distância da célula à rodovia principal (`distancia_principal`), entre outros (Tabela 5.2). O arquivo de vizinhanças é um arquivo texto (Tabela 5.3), gerado por outro programa, que será utilizado para gerar a matriz de proximidade, cuja estrutura foi apresentada no Capítulo 4. No caso desta aplicação, em particular, foi desenvolvido um programa em linguagem C++, utilizando as estruturas de dados de matriz de proximidade disponíveis no TerraLib. Entretanto, qualquer outro programa pode ser utilizado desde que gere um arquivo texto com o conteúdo especificado no dicionário de dados apresentado na Tabela 5.4.

A demanda global e inicial são variáveis globais, isto é, valores absolutos para o autômato como um todo.

TABELA 5.1 Layer Células450 do banco Rondônia.mdb.

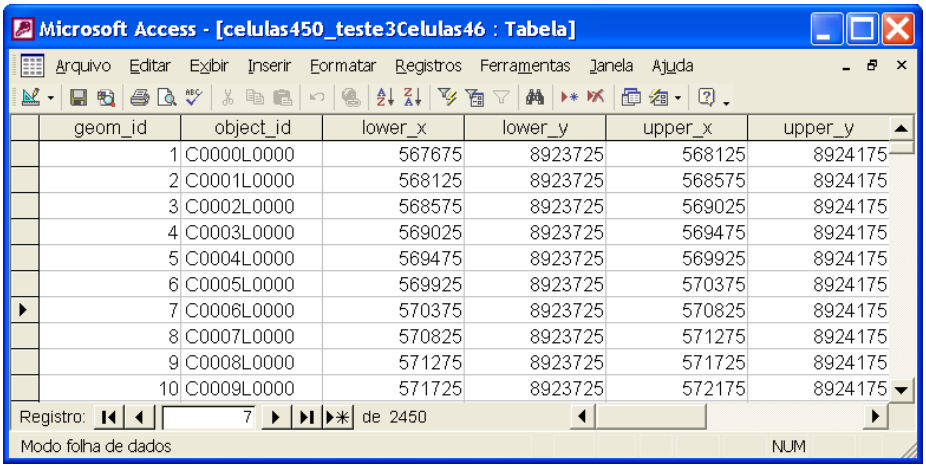


TABELA 5.2 Atributos das células do banco Rondônia.mdb.

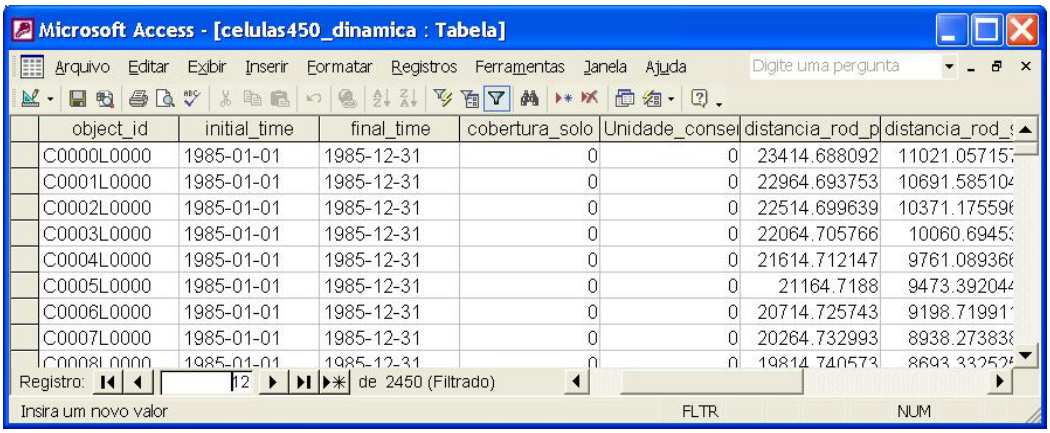


TABELA 5.3 Arquivo de vizinhança.

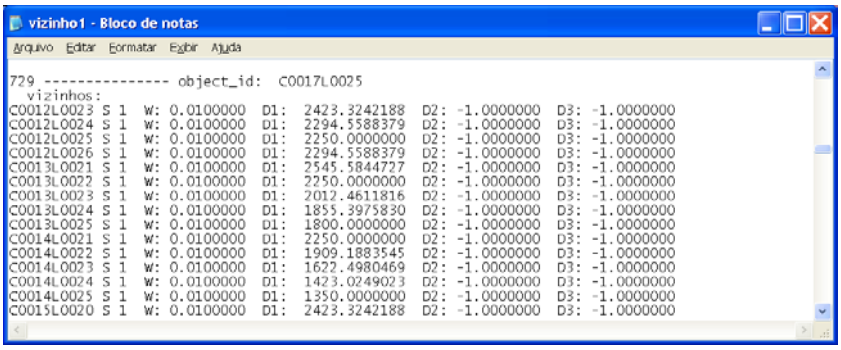


TABELA 5.4 Dicionário de dados dos arquivos do banco de dados rondonia.mdb.

Tabela: Células450Teste3Células 46		
Atributo	Descrição	Tipo / Domínio
geom_id_	Identificador da geometria do objeto	Número inteiro
object_id	Identificador do objeto	String
lower_x	Coordenada x do canto inferior esquerdo	Número inteiro
lower_y	Coordenada y do canto inferior esquerdo	Número inteiro
upper_x	Coordenada x do canto superior direito	Número inteiro
upper_y	Coordenada y do canto superior direito	Número inteiro
col_number	Identficação da coluna da célula	Número inteiro
row_number	Indentificação da linha da célula	Número inteiro
Tabela: Células450Dinâmica		
Atributo	Descrição	Tipo / Domínio
attr_id	Identificador do atributo	
object_id	Identificador do objeto	String
initial_time	Tempo inicial	Data
final_time	Tempo final	Data
cobertura_solo	Cobertura do solo	Número inteiro / 0 ou 1
unidade_ conservação	Unidade de conservação	Número inteiro / 0 ou 1
distancia_ rod_princip	Distância da célula à rodovia principal	Número real
distancia_ rod_secund	Distância da célula à rodovia secundária	Número real
Arquivo: Vizinho1.txt		
Atributo	Descrição	Tipo / Domínio
object_id	Identificador do objeto	string
s	Zona de distância	Número inteiro
w	Peso	Número real
d1	Distância entre as duas células vizinhas	Número real
d2	Distância das células vizinhas à rede de transporte (matriz de proximidade gerada pela rede de transporte)	Número real
d3	Distância das células vizinhas através da rede de transporte (matriz de proximidade gerada pela rede de transporte)	Número real

5.6.2 - Definição dos modos de controle

No início da seção CONTROL são especificados os parâmetros temporais desta simulação. Neste exemplo, serão feitas 10 simulações a partir de "1985", ano a ano (step).

Neste sistema existem dois modos de controle. O modo LOCAL e GLOBAL. No modo LOCAL estão especificadas as condições de fluxo, que têm que ser executadas para cada célula, em cada passo (step) da simulação. No modo global estão as equações que calibram o modelo, atualizam a demanda global e determinam a demanda para o próximo passo.

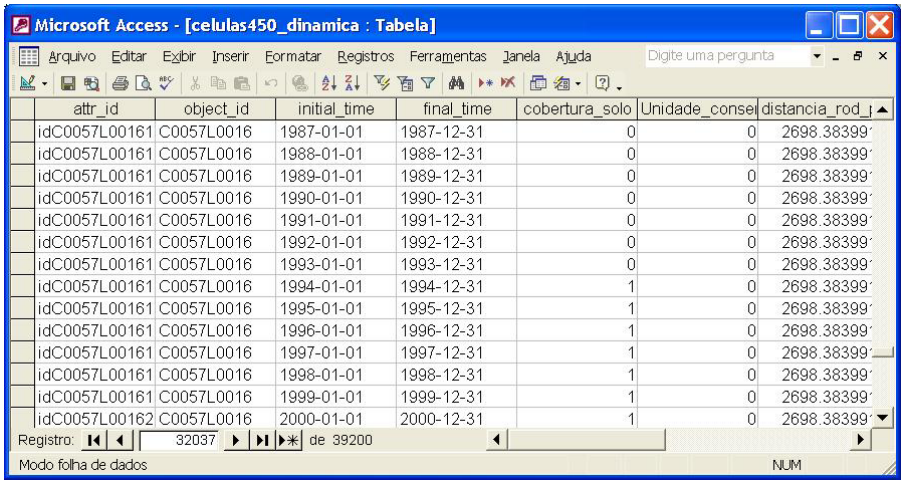
Neste exemplo o sistema transita do modo GLOBAL para o modo LOCAL enquanto houver demanda.

Os resultados gerados são gravados no mesmo banco rondonia.mdb. É gerada uma tabela para cada ano da simulação.

5.6.3 - Resultados

Os resultados das simulações são armazenados na tabela de atributos dinâmicos das células. Por se tratar de um banco de dados temporal, para cada resultado gerado, é registrado o intervalo de tempo (initial_time e final_time). A Tabela 5.5 ilustra o caso da célula C0057L0016 (object_id). No período de 1986 a 1993 a célula não sofreu alteração no atributo cobertura_solo (0 = floresta). Em 1994, a célula tem seu valor de cobertura_solo alterado para 1 (desmatado) e permanece com este valor até o fim das simulações.

TABELA 5.5 Resultado da simulação para uma célula.

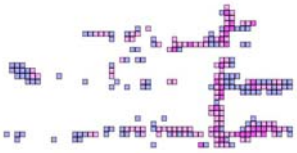
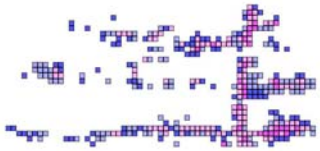
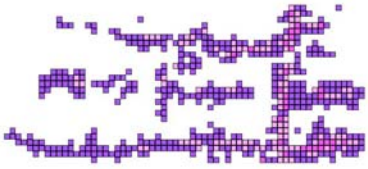
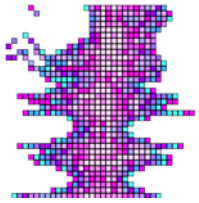
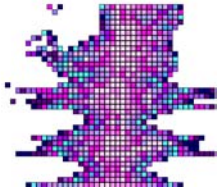


attr_id	object_id	initial_time	final_time	cobertura_solo	Unidade_conse	distancia_rod_
idC0057L00161	C0057L0016	1987-01-01	1987-12-31	0	0	2698.38399
idC0057L00161	C0057L0016	1988-01-01	1988-12-31	0	0	2698.38399
idC0057L00161	C0057L0016	1989-01-01	1989-12-31	0	0	2698.38399
idC0057L00161	C0057L0016	1990-01-01	1990-12-31	0	0	2698.38399
idC0057L00161	C0057L0016	1991-01-01	1991-12-31	0	0	2698.38399
idC0057L00161	C0057L0016	1992-01-01	1992-12-31	0	0	2698.38399
idC0057L00161	C0057L0016	1993-01-01	1993-12-31	0	0	2698.38399
idC0057L00161	C0057L0016	1994-01-01	1994-12-31	1	0	2698.38399
idC0057L00161	C0057L0016	1995-01-01	1995-12-31	1	0	2698.38399
idC0057L00161	C0057L0016	1996-01-01	1996-12-31	1	0	2698.38399
idC0057L00161	C0057L0016	1997-01-01	1997-12-31	1	0	2698.38399
idC0057L00161	C0057L0016	1998-01-01	1998-12-31	1	0	2698.38399
idC0057L00161	C0057L0016	1999-01-01	1999-12-31	1	0	2698.38399
idC0057L00162	C0057L0016	2000-01-01	2000-12-31	1	0	2698.38399

Na Tabela 5.6 são apresentados os resultados da simulação em TerraML (Figura 5.8). Estes resultados foram visualizados através do TerraView, uma ferramenta de visualização geográfica desenvolvida a partir da biblioteca TerraLib (TerraLib 2003).

Através da comparação dos resultados pode-se concluir que o processador TerraML valoriza mais o efeito da vizinhança, do que a acessibilidade (proximidade de rodovias). Na Figura 5.8 pode-se verificar que, para esta simulação, foi atribuído o peso de 0.4 para `acessibilidade1`, e 0.4 para `acessibilidade2`, totalizando 0.8 para acessibilidade de uma forma geral, contra 0.2 para `atratividade`, que incorpora o efeito dos vizinhos. Um dos motivos para as diferenças entre os resultados simulados e a evolução real do desmatamento é o fato da vizinhança das células, para esta aplicação, ter sido construída com base na distância entre as células. O mais adequado, neste caso, seria construir a vizinhança das células com base na rede de transporte. Outro motivo para as diferenças, entre o real e o simulado, está na simplicidade das equações introduzidas nesta aplicação. Um aperfeiçoamento necessário é a definição de novas equações que capturem de forma mais realista a evolução do desmatamento.

TABELA 5.6 Comparação entre os resultados da simulação e o real.

Ano	Real	Simulação
1985		
1988		
1991		
1994		
1997		
2000		

Em suma, os resultados dos experimentos realizados com TerraML não são conclusivos, mas são indicativos da capacidade do ambiente computacional. O ambiente trabalha com uma lógica simplificada, que pode ser refinada com a ampliação, em trabalhos futuros, do número de funções disponíveis.

CAPÍTULO 6
CONCLUSÕES

6.1 - Resultados Alcançados

TerraML foi desenvolvida com o objetivo de diminuir algumas das limitações existentes nos ambientes computacionais para modelagem dinâmica. Algumas destas limitações referem-se à estacionariedade das vizinhanças e à discretização do modelo de mudanças num espaço de estados finito, em função do paradigma de autômato celular clássico. Nestes dois aspectos, TerraML conseguiu avanços.

A primeira contribuição de TerraML é suportar qualquer vizinhança que possa ser representada por uma matriz de proximidade. Numa matriz de proximidade, uma célula pode ter qualquer número de vizinhos. E estes vizinhos podem variar não só em número, mas também em direção, ou seja, podem variar também na sua configuração espacial, isto é, podem apresentar formas diferentes de célula para célula. De uma forma geral, aplicações de geoprocessamento baseadas em autômatos celulares definem vizinhanças de configuração fixas, dos tipos Moore ou Von Neumman, com algumas variações no número de zonas de distâncias.

A segunda contribuição está em suportar um modelo de mudanças híbrido, com elementos contínuos e discretos, num espaço de estados infinito. TerraML utiliza os conceitos de autômatos celulares híbridos para representar o espaço e as transições de estado que nele ocorrem. Uma das vantagens desta abordagem de implementação está na flexibilidade para combinar elementos discretos e contínuos de forma integrada. Outra vantagem é a facilidade de implementação computacional, uma vez que existe uma relação direta entre os elementos do autômato híbrido, como as condições de fluxo e de mudança, e as estruturas de controle de repetição (*loops*) e comandos de seleção (*ifs*), existentes em qualquer ambiente de desenvolvimento de software.

Por fim, o fato de TerraML fazer parte da TerraLib traz, pelo menos, dois benefícios imediatos . O primeiro deles é o fato de contar com o respaldo de uma equipe de desenvolvimento com mais de 20 anos de experiência no desenvolvimento de aplicações geográficas. A segunda, é que um software com o “padrão de qualidade”

TerraLib tem que atender requisitos rigorosos de eficiência e interoperabilidade devido ao ambiente colaborativo da biblioteca. Algumas das aplicações TerraLib são: conversor de formatos, analisador de dados em saúde publica, gerenciamento municipal (TerraCrime, TerraEmTransito) e modelagem dinâmica (TerraML).

6.2 - Futuros Trabalhos

Uma tendência em sistemas sofisticados de computação é o uso de agentes. Agentes são componentes de software que podem ser dotados de conhecimentos e se adaptar a diferentes ambientes. O uso de agentes é uma das possibilidades de aperfeiçoamento do TerraML. Para isto, devem ser investigadas as arquiteturas de agentes, plataformas e "ferramentas" computacionais para a implementação de Sistemas Multi-Agentes, definir de estratégias de interação e negociação para a cooperação e a resolução de conflitos, manutenção de coerência em ambientes abertos e domínios de aplicação.

Uma questão crucial para sistemas de modelagem dinâmica é o modelo matemático em si. Em geral, os modelos são representações simplificadas da realidade, construídas muito mais sob as restrições do domínio da solução, do que sob as características das variáveis e leis que governam o comportamento do fenômeno que o modelo descreve. A evolução da computação tem propiciado o surgimento de tecnologias que permitem introduzir aperfeiçoamentos nas formas de tratar e embutir modelos em software. Um trabalho de investigação ou até mesmo a proposta de um "novo" modelo para mudança de uso do solo impulsionaria o desenvolvimento de novas funcionalidades e estruturas de dados e processos para o ambiente TerraML.

Um outro aperfeiçoamento, não para o ambiente TerraML em particular, mas para sistemas de “vida artificial” em geral, é a exploração de técnicas e arquiteturas de processamento paralelo. O modelo celular é o ambiente ideal para que o paralelismo seja implementado, uma vez que células guardam poucas informações, estão altamente conectadas e realizam tarefas muito simples. Entretanto, as células são sempre em grande número, o que degrada o desempenho destes sistemas. Neste aspecto, o processamento paralelo pode ser a solução para tornar estes sistemas mais eficientes. Como no caso dos agentes, mencionado anteriormente, técnicas e

arquiteturas de computação paralela devem ser estudas para se verificar qual a solução ideal para um sistema de modelagem dinâmica.

REFERÊNCIAS BIBLIOGRÁFICAS

Alves, D. S. Space-time dynamics of deforestation in Brazilian Amazônia. **International Journal of Remote Sensing,** v. 23, n. 14, p. 2903-2908, 2002.

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

Batty, M. Geocomputation using cellular automata. In: Openshaw, S.; Abrahart, R. J.(ed.) **Geocomputation.** London: Taylor&Francis, 2000. p. 95-126.

Booch, G.; Rumbaugh, J.; Jacobson, I. **The unified modeling language user guide**. New York: Addison Wesley, 1998. 478 p.

Burrough, P. Dynamic modelling and geocomputation. In: Longley, P.A.; Brooks, S.M.; McDonnel, R.; MacMillan, B. (ed.). **Geocomputation:** a primer**.** London: John Wiley & Sons, 1998. p. 165-190.

Burrough, P.; McDonnel, R. **Principles of geographical information systems.** Oxford: Oxford University Press,1998. 320 p.

Câmara, A. S. Spatial simulation modelling. In: Fisher, M. (ed.).**Spatial analytical perspectives on GIS**. London: Taylor & Francis, 1996. p. 213-218.

Câmara, G.; Souza, R.C.M.; Pedrosa, B. M.; Vinhas, L.; Monteiro, A.M.V.; Paiva, J.A.; Carvalho, M.T.; Gatass, M. TerraLib: Technology in Support of GIS Inovation. In: GEOINFO 2000 - Workshop Brasileiro de Geoinformação, 2., 2000, São Paulo. **Anais…** São Paulo, [s.n], 2000. p. 126-133.

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

Couclelis, H. Space, time, geography. In: Longley, P.; Goodchild, M.; Maguire, D.; Rhind, D. (ed.). **Geographical information systems.** New York: John Wiley & Sons, 1999. p. 29-38.

Edelweiss, N.; Oliveira, J. P. M. Modelagem de aspectos temporais de sistemas de Informação. In: **IX Escola de computação**, Recife: Universidade Federal de Pernambuco, 1994.

Elmasri, R.; Navathe, S. B. **Fundamentals of database systems.** Redwood, CA: Addison-Wesley. 2000. 873 p.

Engelen, G.; White, R.; Uljee, I. Exploratory Modelling of Socio-Economic Impacts of Climatic Change. In: Maul, G.A. (ed.). **Climate Change in the Intra-America's Sea**, London: Edward Arnold, 1993. p. 306-324.

Engelen, G.; White, R.; Uljee, I.; Drazan, P. Using cellular automata for integrated modelling of socio-environmental systems. **Environmental Monitoring and Assessment**, v. 34, n.2, p. 203-214, 1995.

Engelen, G., Uljee, I.; White, R. **Vulnerability assessment of low-lying coastal areas and small islands to climate change and sea level rise -** fhase 2: case study St. Lucia. Maastricht: The Netherlands: UNEP - CAR/RCU, RIKS Publication. Report & SimLucia user manual.

GNU's Not Unix. **The GNU Project and the Free Software Foundation (FSF)**.Boston: Free Software Foundation. Disponível em: <www.gnu.org>. Acesso em: 23 May 2002.

Henzinger, T. A. The Theory of Hybrid Automata. In: Symposium on Logic in Computer Science, 11.,1996, New Brunswick, NJ . **Proceedings...**New Brunswick: IEEE Computer Society Press, 1996, p. 278-292.

Hornsby, K.; Egenhofer, M. J. Qualitative Representation of Change. Spatial Information Theory: A Theoretical Basis for GIS. In: International Conference on Spatial Information Theory, 3.,1997, Berlim. **Proceedings…** Berlim: Lecture Notes in Computer Science, Springer-Verlag. 1997. p.15-33.

Lambin, E. F. **Modeling deforestation processes -** a review, . Luxembourg: European Commission. 1994. Trees series B: Research Report.

Openshaw, S. GeoComputation. In: Openshaw, S.; Abrahart, R. J.(ed.). **Geocomputation.** London: Taylor&Francis, 2000. p. 1-31.

O'Sullivan, D. Toward micro-scale spatial modeling of gentrification. **Journal of Geographical Systems,** v. 4, n. 3, p. 251-74, 2002.

Parent, C.; Spaccapietra, S.; Zimányi, E. Spatio-temporal conceptual models: data structures + space + time. In: ACM Symposium on Advances in Geographic Information Systems, 7., Kansas City - MO,.**Proceedings...** Kansas: ACM, 1999. p. 26-33.

Peuquet, D. Making space for time: issues in space-time data representation. **GeoInformatica**, v. 5, n. 1, p. 11-32, 2001.

Reis, E. J.; Margulis, S. Options for slowing Amazon jungle clearing. In: Dornbusch, R.; Poterba, J. M.(ed.). **Global warming:** economic policy responses. Cambridge: The MIT Press, 1991, p. 335-375.

Research Institute for Knowledge Systems. **RIKS - f**aculty of spatial science. Utrecht: University of Utrecht. Disponível em: <www.riks.nl>. Acesso em: May 2001.

Roy, G. G.; Snickars, F. Citylife: a study of cellular automata in urban dynamics. In: Fisher, M. **Spatial analytical perspectives on GIS.** London: Taylor & Francis, 1996, p. 213-218.

Santos, M. **A natureza do espaço: técnica e tempo, razão e emoção**. São Paulo: EDUSP.2002. 392 p.

Sipper, M. The emergence of cellular computing. **IEEE Computer,** v. 32, n. 7, 1999, p.18-26.

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

Soares Filho, B. S. **Modelagem dinâmica de paisagem de uma região de fronteira de colonização amazônica.** Tese de doutorado. Escola Politécnica - Universidade de São Paulo, São Paulo, 1998.

Instituto Nacional de Pesquisas Espaciais.Divisão de Processamento de Imagens (INPE/DPI). **TerraLib**. Disponível em : <http://terralib.dpi.inpe.br>. Acesso em: julho 2003.

van Deursen, W. P. A. **Geographical information systems and dynamic models.** Tese de doutorado. Faculty of Spatial Sciences, Rotterdan - The Netherlands: University of Utrecht, 1995.

van Gurp, J., Bosch, J. On the implementation of finite state machines. In: Annual lasted International Conference on Software Engineering and Applications, 3.,1999, Scottsdale. **Proceedings…** Scottsdale - Arizona: 1999. p. 172-178.

Wolfran, S. Statistical mechanics of cellular automata. **Reviews of Modern Physics**, v.55, p. 601-644. July 1983.

World Wide Web Consortium. **W3C**. Disponível em <www.w3.org>. Acesso em: May 2003.

White, R.; Engelen,G. Cellular automata as the basis of integrated dynamic regional modelling. **Environment and Planning B: Planning and Design**, v. 24, p.165-174.1997.

Worboys, M. F. **GIS -** a computing perspective. Bristol - PA: Taylor & Francis Inc, 1995. 376 p.

Zipf, A.; Krüger, S. TGML - Extending GML by temporal constructs - a proposal for a spatiotemporal framework in XML. In: . ACM International Symposium on Advances in Geographic Information Systems, 7. Atlanta. **Procedings….** Atlanta: ACM, 2001. p. 94-99.

APÊNDICE A

DTD completo da Linguagem TerraML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by Bianca Pedrosa (Particular) -->
<!--DTD generated by XMLSPY v2004 rel. 2 U (http://www.xmlspy.com)-->
<!ELEMENT cellprocessor (input, control)>
<!ATTLIST cellprocessor
	author CDATA #REQUIRED
	date CDATA #REQUIRED
	model CDATA #REQUIRED
>
<!ELEMENT condition EMPTY>
<!ATTLIST condition
	attribute CDATA #REQUIRED
	op CDATA #REQUIRED
	value CDATA #REQUIRED
>
<!ELEMENT control (mode+, transition)>
<!ATTLIST control
	initime CDATA #REQUIRED
	intervals CDATA #REQUIRED
	step CDATA #REQUIRED
	timeUnit  (YEAR | MONTH | DAY | HOUR )  #REQUIRED
>
<!ELEMENT database EMPTY>
<!ATTLIST database
	host CDATA #REQUIRED
	path CDATA #REQUIRED
	name CDATA #REQUIRED
	user CDATA #IMPLIED
	pass CDATA #IMPLIED
>
<!ELEMENT expander EMPTY>
<!ATTLIST expander
	attribute CDATA #REQUIRED
	column CDATA #REQUIRED
	demand CDATA #REQUIRED
>
<!ELEMENT fuzzyL EMPTY>
<!ATTLIST fuzzyL
	attribute CDATA #REQUIRED
	column CDATA #REQUIRED
	alpha CDATA #REQUIRED
	beta CDATA #REQUIRED
>
<!ELEMENT global EMPTY>
<!ATTLIST global
	name CDATA #REQUIRED
	value CDATA #REQUIRED
>
<!ELEMENT input (database, layer, table, neighborhood, temporal, global+)>
<!ELEMENT layer EMPTY>
<!ATTLIST layer
	name CDATA #REQUIRED
	layerid CDATA #REQUIRED
>
<!ELEMENT localMean EMPTY>
<!ATTLIST localMean
	attribute CDATA #REQUIRED
	column CDATA #REQUIRED
>
<!ELEMENT mode (fuzzyL*, localMean?, product, expander?)>
<!ATTLIST mode
	name CDATA #REQUIRED
>
<!ELEMENT neighborhood EMPTY>
<!ATTLIST neighborhood
	name CDATA #REQUIRED
	zones CDATA #REQUIRED
>
<!ELEMENT pair EMPTY>
<!ATTLIST pair
	attribute CDATA #REQUIRED
	weight  CDATA  #REQUIRED
>
<!ELEMENT product (pair+)>
<!ATTLIST product
```

```
    attribute CDATA #REQUIRED
>
<!ELEMENT table EMPTY>
<!ATTLIST table
    name CDATA #REQUIRED
    columns CDATA #REQUIRED
    lines CDATA #REQUIRED
>
<!ELEMENT temporal EMPTY>
<!ATTLIST temporal
    name CDATA #REQUIRED
    attribute CDATA #REQUIRED
    period CDATA #REQUIRED
    timeunit (YEAR | MONTH | DAY | HOUR ) #REQUIRED
>
<!ELEMENT transition (condition)>
<!ATTLIST transition
    from CDATA #REQUIRED
    to CDATA #REQUIRED
>
```

APÊNDICE B

Código Fonte

Obs: Esta documentação destina-se a fornecer uma visão geral da abordagem de implementação. Os arquivos de cabecalho (.h) estão completos. Dos arquivos fonte (.cpp) foram selecionadas as funções mais significativas do sistema.

```
#ifndef TeCellA_H
#define TeCellA_H

//Includes
//---------------------------TerraLib
#include <TeCellAlgorithms.h>
#include <TeAdoDB.h>
#include <TeLayer.h>
#include <TeGeometry.h>
#include <TeSelectedObject.h>
#include <TeImportExport.h>
#include <TeTime.h>
#include <TeInitRasterDecoders.h>
#include <TeDecoderDatabase.h>
#include <TeRasterRemap.h>
#include <AuxiliarAlgorithms.h>
//----------------------------Parser
#include <dom/DOM_Node.hpp>
#include <dom/DOM_Element.hpp>
//----------------------------TerraML
#include "TeProxMtx.h"
#include "controlmode.h"
#include "bicell.h"

class TeCellAutomata
{
public:
    TeCellAutomata(char *x);
    void builder (DOM_Node& aNode);
    void buildsection (DOM_Node& aNode);
    int init( string, string , string , string , string, int, string, char *, int);
    void inputdata (DOM_Node& aNode);
    void inputcontrol (DOM_Node& aNode);
    void inputtime (DOM_Node& aNode);

    void printcells();
    void printDynCells();
    void print_neighbors();
    void printcontrols();

    bool createneighbors(char *arquivo, int zonas);
    bool createDynAttribute();

    bool execute();
    bool process(ControlMode, int t);
    void calculate (Flow *f, int t);
    void fuzzyL(FuzzyL *f, int t);
    void localMean(LocalMean *f,int t);
    void sum (Product *f,int t) ;
    void expander(Expander *f, int t);
    double sort(string attribute, double demand, int t);
    void transit(string attribute, string column, double threshold, int t);
    ControlMode *getMode(string s);
private:
    BiCellSet   cells_;
    TeProxMatrix  neighborhood_;
    TeTimeSequence *timeseq_;
    ModeSet       modes_;
    TransitionSet transitions_;

};

#endif
```

```
#ifndef BICELL_H
#define BICELL_H

#include <vector>
#include <map>
#include <TeGeometry.h>
#include <TeAttribute.h>

typedef vector<double> doubleRow;
typedef vector<doubleRow> doubleTable;
class BiCell : public TeCell
{
private:
    TeTableRow          attribute_;     // Modeling Attributes - TerraML
    doubleTable         dynAttribute_;    // Dynamic Attributes - TerraML

public:
    BiCell(string o) {objectId_=o;}

    BiCell (string o, int c, int l)
    {   objectId_=o;
        this->column(c);
        this->line(l);
    }

    //! Adds modeling attribbutes to the  cell
    void setAttributes(TeTableRow s) { attribute_ = s;}

    //! Gets modeling attributes (the entire row)
    TeTableRow getAttributes () {return attribute_;}

    //! Adds modeling attribbutes to the  cell
    void attribute(string s) { attribute_.push_back(s);}

    //! Return the i-th attribute of the cell
    string attribute(int i) {return attribute_[i];}

    //! Adds new line of dynamic attributes to the  cell
    void dynAttribute(doubleRow s) { dynAttribute_.push_back(s);}

    //! Return the dynamic attributes of the cell
    doubleRow dynAttribute(int i) {return dynAttribute_[i];}

    //da
    void dynAtt(int l, int c, double value)
    {   dynAttribute_[l][c]=value;  }

    double dynAtt(int l, int c)
    {return dynAttribute_[l][c];}
};
```

```
//!  A class for handling sets of BiCells
class BiCellSet: public  TeGeomComposite<BiCell>
{
    double resX_; //!< the X resolution of a set of cells
    double resY_; //!< the Y resolution of a set of cells

    TeAttributeList    attributes_;
    TeAttributeList    dynAttributes_;

public:

    //! operador de copia from TeCell
    operator = ( TeCellSet cells)
    {
        cout << "copiando TeCell em BiCell..."<< endl;
        TeCellSet::iterator cells_it=cells.begin();
        while (cells_it != cells.end())
        {
            BiCell bcell((*cells_it).objectId(),(*cells_it).column(),(*cells_it).line());
            this->add(bcell);
            cells_it++;
        }
    }


    //! Returns the X resolution of a cell set
    double resX ()
    { return resX_; }

    //! Returns the Y resolution of a cell set
    double resY ()
    { return resY_; }

    //! Sets the X resolution of a cell set
    void resX (double reX)
    { resX_ = reX; }

    //! Sets the Y resolution of a cell set
    void resY (double reY)
    { resY_ = reY; }

          //! Sets the attribute list of a cell set
    void attributes(TeAttributeList t) {attributes_=t;}

    void dynattributes(TeAttributeList t) {dynAttributes_=t;}

    string attributes(int n) {return attributes_[n].rep_.name_;}

    string dynAttributes(int n) {return dynAttributes_[n].rep_.name_;}

    int  attribute(string s) //! retorna o nro da coluna do atributo s
    {
        int j  = 0;
        TeAttribute  attr;
        TeAttributeList::iterator att_it = attributes_.begin();
        while ( att_it != attributes_.end())
        {
            if ((*att_it).rep_.name_ == s)
            {
                return j;
            }
            j++;
            att_it++;
        }
        return -1;
    }
```

```
    int  dynAttribute(string s) //! retorna o nro da coluna do atributo s
    {
        int j  = 0;
        TeAttribute  attr;
        TeAttributeList::iterator att_it = dynAttributes_.begin();
        while ( att_it != dynAttributes_.end())
        {
            if ((*att_it).rep_.name_ == s)
            {
                return j;
            }
            j++;
            att_it++;
        }
        return -1;
    }

    //! Sets the dynamic attribute list of a cell set
    void dynAttributes(TeAttributeList t) {dynAttributes_=t;}

    //! Returns the attribute list of a cell set
    TeAttributeList attributes() {return attributes_;}

    //! Returns the dynamic attribute list of a cell set
    TeAttributeList dynAttributes() { return dynAttributes_;}


};

#endif
```

```
#ifndef TEPROXMTX_H
#define TEPROXMTX_H
#include <map>

struct ProxMatrixInfo {
    float weight_;
    int slice_;
    float d1_,d2_,d3_; //d1=centroidDistance, d2=NetDistance, d3=NetMinPath

    ProxMatrixInfo (): weight_(0.00), slice_(0), d1_(0.00),  d2_(0.00), d3_(0.00){}

    ProxMatrixInfo (float w,int s,float d1,float d2,float d3):
                            weight_(w), slice_(s), d1_(d1),  d2_(d2), d3_(d3){}

    ProxMatrixInfo (const ProxMatrixInfo& c): weight_(c.weight_), slice_(c.slice_), d1_(c.d1_),
                            d2_(c.d2_), d3_(c.d3_){}

    ProxMatrixInfo& operator = (const ProxMatrixInfo& x)
    {   weight_= x.weight_;
        slice_= x.slice_;
        d1_=x.d1_;
        d2_=x.d2_;
        d3_=x.d3_;
        return *this;
    }
    void getProxMatrixInfo()
    {   cout << " weight = " <<weight_ << " slice = "<< slice_ << " d1 =" << d1_ << endl;

    }
    bool empty()
    {   if (slice_>=0) return true;
        return false;
    }
    float weight() { return weight_;}
};

typedef map<TeCell*, ProxMatrixInfo> stringRow;
typedef map<string, stringRow> sparceMatrix;

class TeProxMatrix: public TeSingleton<TeProxMatrix>
{
public:
    void insertNeighbour (
        const string& o1, TeCell* o2,
        int slice, float weight, float d1, float d2, float d3)
    {
        ProxMatrixInfo pm ( weight, slice, d1, d2, d3);
        matrix_ [o1] [o2] = pm;
    }

          stringRow getNeighborhood ( string o1 )
    {
        return matrix_ [o1];
    }

    ProxMatrixInfo getANeighbor ( string o1, TeCell* o2 )
    {
        return matrix_ [o1] [o2];
    }
protected:
    sparceMatrix matrix_;

};

#endif
```

```
#ifndef CTRLMODE_H
#define CTRLMODE_H

#include <string>

using namespace std;

class Flow
{
    string attr_;
    string math_;
public:
    Flow(string a, string m): attr_(a), math_(m){}
    string attr()   {   return attr_;}
    string math() {   return math_;}

};
typedef vector <Flow*> FlowSet;

class Condition
{   string attr_, value_, oper_;
public:
    Condition(string a, string v, string o): attr_(a), value_(v), oper_(o){}
    string attr()   {   return attr_;}
    string oper()  {   return oper_;}
    string value() {   return value_;}

};

class ControlMode
{   string          name_;
    FlowSet             flows_;

public:
    //ControlMode();
    ControlMode():name_(""){}
    ControlMode(string s) {name_=s;}
    string name() {return name_;}
    FlowSet flows() {return flows_;}
    void add(Flow *aFlow) {flows_.push_back(aFlow); }
    void print()
    {   cout << name_ << endl;
        int n=flows_.size();
        for (int i=0;i<n;i++)
            cout << (*flows_[i]).attr() <<endl;
    }
};
typedef vector<ControlMode> ModeSet;

class Transition
{   ControlMode       *fromode_,*tomode_;
    vector<Condition >  jumps_;

public:
    Transition (ControlMode* f, ControlMode* t): fromode_(f),tomode_(t){}
    void add (Condition aCondition) {jumps_.push_back(aCondition);}
    void print()
    {   cout << (*fromode_).name() << ' ' << (*tomode_).name() << endl;
        for (int i=0;i<jumps_.size();i++)
                    cout << jumps_[i].attr() << ' ' << jumps_[i].oper()<< ' '
                    <<jumps_[i].value()<<endl;
    }
};

typedef vector <Transition> TransitionSet;
```

```
class FuzzyL : public Flow
{   string column_;
    double alpha_;
    double beta_;
public:
    FuzzyL (string a, string m,string c, double aa, double b):Flow(a,m), column_(c), alpha_(aa),
beta_(b){}
    string column() {return column_; }
    double alpha()  {return alpha_;    }
    double beta() {   return beta_; }
};

class LocalMean : public Flow
{   string column_;
public:
    LocalMean (string a, string m, string c):Flow(a,m), column_(c){}
    string column() {return column_; }
};

class Expander : public Flow
{   string column_;
    double demand_;
public:
    Expander (string a, string m, string c, double d):Flow(a,m), column_(c), demand_(d){}
    string column() {return column_; }
    double demand() {   return demand_; }
};
#endif
```

```
TeCellAutomata :: TeCellAutomata (char *xmlFile)
{   DOM_Node doc_ = xmlParser(xmlFile);
    builder(doc_);
}

void TeCellAutomata::builder(DOM_Node& aNode)
{
            if (is_section( aNode.getNodeName().transcode() ))
            {   buildsection(aNode);
                aNode = aNode.getNextSibling();
            }
            else
            {   //  Test for the presence of children
                DOM_Node child = aNode.getFirstChild();
                while( child != 0)
                {
                    builder(child);
                    child = child.getNextSibling();
                }
            }
}

void TeCellAutomata::buildsection (DOM_Node& aNode)
{
      //  Test for the presence of children
    if (!strcmp(aNode.getNodeName().transcode(),"input"))
        inputdata(aNode);
    if (!strcmp(aNode.getNodeName().transcode(),"control"))
        inputcontrol(aNode);
   if (!strcmp(aNode.getNodeName().transcode(),"simulation"))
        inputtime(aNode);
    else
    {   DOM_Node child = aNode.getFirstChild();
        while( child != 0)
        {   child = child.getNextSibling();
        }
    }
}

int TeCellAutomata::init( string host, string dbname, string user, string pass, string layername, int
layerid, string tablename, char *neighborsname,int zonas)
{
    cout << "lendo banco " << host << ' ' << dbname << "...." <<endl ;
          // Opens a connection to a database accessible though ADO
        TeDatabase* ado_db = new TeAdo();
         if (!ado_db->connect(host,user,pass,dbname,0))
        {
             cout << "Error: " << ado_db->errorMessage() << endl;
             exit(0);
         }
    TeCellSet cells;
    if (!ado_db->loadCellSet (layerid, layername, "", cells))
    {

        cout << "\tLayer de entrada inexistente: " << layername << endl;
         ado_db->close();
         exit(0);
    }
    cells_ = cells;
    TeTable    cellstable;
    if (!ado_db->loadTable (tablename, cellstable))
    {
         cout << "\tTable de entrada inexistente: " << tablename << endl;
         ado_db->close();
         exit(0);
    }
    assignAttr(cells_,cellstable);         //adiciona atributos nas celulas
    createneighbors(neighborsname,zonas);  //gera matriz de proximidade
    return 1;
}
```

```
bool TeCellAutomata::execute()
{
    createDynAttribute();
    for (int t=0; t<(*timeseq_).num_steps(); t++)
    {   cout << "T " << t<< endl;
        process((*modes_.begin()),t); //generalizar - assumi 1o modo local
        result(cells_,t);
    }

    return true;
}


bool TeCellAutomata::process (ControlMode Local, int t)
{
        FlowSet flows= Local.flows();
        int n=flows.size();
        for (int i=0;i<n;i++) //calcula todos os flows deste ControlMode
        {   cout << "flow "<< i << endl;
            calculate(flows[i],t);
        }
        return true;
}


void TeCellAutomata::fuzzyL(FuzzyL  *f,int t)
{   cout << "calculating a fuzzyLow...\n";
    int attcol=cells_.attribute((*f).column());
    double alpha=(*f).alpha();
    double beta=(*f).beta();
    BiCellSet::iterator cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {   double x=atof((*cells_it).attribute(attcol).c_str());
        double value = (1 / (1 + ( alpha * pow( x - beta,2)))) ;
        (*cells_it).dynAtt(t, cells_.dynAttribute((*f).attr()), value);
        cells_it++;
    }
}


void TeCellAutomata::localMean(LocalMean *f,int t)
{   cout << "calculating a local mean...\n";
    BiCellSet::iterator cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {   string attribute = (*f).column();
        double result=0.00;
        string      o1 = (*cells_it).objectId();
        for (BiCellSet::iterator it=cells_.begin(); it!= cells_.end(); it++)
        {   ProxMatrixInfo pmi =TeProxMatrix::instance().getANeighbor(o1,it);
            if (pmi.empty())
            {   if (t==0)
                {   if (atof((*it).attribute(cells_.attribute(attribute)).c_str()) > 0)
                        result+=pmi.weight();
                }
                else
                {   if ((*it).dynAtt(t-1,cells_.dynAttribute(attribute)) > 0)
                        result+=pmi.weight();
                }
            }
        }
        (*cells_it).dynAtt(t, cells_.dynAttribute((*f).attr()), result);
        cells_it++;
    }
}
```

cellularAutomata.cpp

```
void TeCellAutomata::expander(Expander *f, int t)
{   int globalPotential =0;
    string attribute=(*f).attr();
    double demand=(*f).demand();
    string column=(*f).column();
    BiCellSet::iterator cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {
        if ((*cells_it).dynAtt(t, cells_.dynAttribute(column))>0)
            globalPotential++;
        cells_it++;
    }
    cout << "total de celulas com potencialpara mudanca: " << globalPotential << endl;
    double threshold = 0.00;
    if (globalPotential>demand)
        threshold = sort(column, demand, t);
    transit(attribute, column, threshold, t);
}


double TeCellAutomata::sort(string attribute, double demand, int t)
{
    cout << "expanding..." << endl;
    //gera uma lista de potenciais para poder ordenar
    list<double> myList;
    BiCellSet::iterator cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {   double value = (*cells_it).dynAtt(t, cells_.dynAttribute(attribute));
       myList.push_back(value);
        cells_it++;
    }
    // ordena potenciais e obtem limiar
    myList.sort();
    double threshold;
    list<double> :: iterator it=myList.end();
    for(int i=0;i<demand;i++)
    {   threshold=(*it);
       it--;
    }
    return threshold;
}


void TeCellAutomata:: transit(string attribute, string column, double threshold, int t)
{
    double value;
    BiCellSet::iterator cells_it=cells_.begin();
    cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {
        if ((*cells_it).dynAtt(t, cells_.dynAttribute(column))>=threshold)
            value=1;
        else
                value=atof( (*cells_it).attribute(cells_.attribute(attribute)).c_str() );

        (*cells_it).dynAtt(t, cells_.dynAttribute(attribute), value);

        cells_it++;
    }
}
```

main.cpp

```
int main( int argc, char ** argv )
{
    TeCellAutomata ca(argv[1]);
    ca.execute();
    return 1;
}
```