Ministério da
Ciência, Tecnologia e Inovação

CellDB: UMA ARQUITETURA PARA SUPORTE A PROBLEMAS DE MODELAGEM AMBIENTAL EM GRANDE ESCALA

Gilberto Ribeiro de Queiroz

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Gilberto Câmara e Dr. Antônio Miguel Vieira Monteiro.

INPE
São José dos Campos
2012

Ficha será revisada pelo SID.

Dados Internacionais de Catalogação na Publicação

Queiroz, Gilberto Ribeiro de.
Cutter CellDB: UMA ARQUITETURA PARA SUPORTE A PROBLEMAS DE MODELAGEM AMBIENTAL EM GRANDE ESCALA / Gilberto Ribeiro de Queiroz. - São José dos Campos: INPE, 2013.
i + 0p. ; (aa/bb/cc/dd-TDI)

Doutorado em Computação Aplicada - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2013.
Orientador: Gilberto Câmara, Antônio Miguel Vieira Monteiro.

1. GIS. 2. DATABASE. 3. SOFTWARE ARCHITECTURE. 4. ENVIRONMENT MODELLING. 5. NOSQL.
I. Título.

CDU

FOLHA DE APROVAÇÃO

CONFECCIONADA PELO SPG E INCLUÍDA PELO SID.

“Lidar com a natureza finita dos atuais computadores é uma arte que requer paciência infinita”.

K. MULMULEY

À minha querida esposa Cássia e
à jovem Manu.

AGRADECIMENTOS

Primeiramente, gostaria de expressar minha admiração e meus sinceros agradecimentos aos orientadores Dr. Gilberto Câmara e Dr. Antônio Miguel Vieira Monteiro, pelos ensinamentos, paciência e atenção com que me orientaram neste trabalho. Também, gostaria de agradecer a oportunidade que me deram de participar desde o início da equipe de desenvolvimento da TerraLib e por acreditarem que de alguma forma eu poderia contribuir para este projeto.

Meu muito obrigado ao grande colaborador e amigo Dr. Pedro Ribeiro de Andrade pelas inúmeras sugestões e experiências compartilhadas ao longo deste trabalho.

Ao Juan, meu grande parceiro no desenvolvimento tecnológico da TerraLib.

Ao Lauro, pelos inúmeros conselhos ao longo dessa caminhada.

À Karine, amiga desde os tempos de universidade, que nunca me deixou desanimar ao longo desta caminhada.

À Lúbia, por me blindar das atividades do projeto TerraLib nos últimos meses da tese.

Ao amigo de sala, Emiliano, pela enorme paciência e pelo puff, providenciado em um ótimo momento.

À Marisa, pelos inúmeros conselhos e incentivos.

Ao Tkörting, novo integrante do Time TerraLib.

Aos amigos do time de desenvolvimento da Funcate, Fred, Douglas, Eric, André e Felipe.

E a todos que de alguma forma contribuíram nesta caminhada.

RESUMO

Nos últimos anos, o volume de dados disponíveis para os pesquisadores realizarem simulações computacionais que representem dinâmicas espacialmente explícitas vem aumentando consideravelmente. Novas tecnologias têm possibilitado a aquisição de dados com resoluções cada vez mais finas e com maior frequência temporal, que são armazenados nos mais diversos formatos. No entanto, grande parte das ferramentas de modelagem dinâmica espacial possue limitada capacidade de manipulação de grandes volumes de dados. Devido a esta limitação, muitos modeladores degradam a resolução espacial do modelo ou reduzem a área de estudo de forma a diminuir os custos computacionais, para tornar o estudo viável. No entanto, para que essas ferramentas possam ser usadas amplamente pela comunidade científica, neste novo cenário, é necessário que elas sejam capazes de lidar com grandes massas de dados. Ao mesmo tempo, existe um grande debate sendo realizado no campo de banco de dados sobre a concepção de novas tecnologias voltadas a nichos mais específicos de aplicação. Partindo deste cenário, esta tese procura contribuir para os estudos de modelagem ambiental integrada, através da proposta de uma arquitetura de software voltada aos requisitos de acesso e gerenciamento de dados das ferramentas de modelagem dinâmica espacial. Os resultados obtidos, a partir de protótipo desenvolvido com base nesta arquitetura, demonstram a sua capacidade em tratar volumes de dados consideráveis em ambiente de computador pessoal.

CellDB: AN ARCHITECTURE FOR DATA ACCESS IN LARGE SCALE ENVIRONMENTAL MODELING BASED ON CELLULAR SPACES

ABSTRACT

In recent years, the volume of data available to researchers performing computer simulations that represent spatially explicit dynamics has increased considerably. New technologies have made it possible to acquire data with increasingly fine resolutions and higher temporal frequency, which are stored in various formats. However, most of simulation tools have limited capacity for handling large volumes of data. Due to this limitation, researchers have to degrade the model resolution or must reduce the study area in order to lower the computational costs, to make the study feasible. However, in order for these tools to be widely used by the scientific community, in this new scenario, they must be able to handle massive datasets. At the same time, in the database field, a debate has been growing about systems that are more responsive to new demands, mainly for non-conventional applications. Based on this scenario, this work contributes to the integrated environmental modeling studies, by proposing a software architecture oriented towards the data management and access requirements spatial dynamics modeling tools. The results of our prototype, based on this architecture, demonstrate its ability to handle a substantial volume of data in a personal computer environment.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

INPE	Instituto Nacional de Pesquisas Espaciais
OGC	Opengeospatial Consortium
UML	Unified Modeling Language
WCS	Web Coverage Service
WFS	Web Feature Service

LISTA DE SÍMBOLOS

GB	Gibabyte
MB	Megabyte

SUMÁRIO

Pág.

1 INTRODUÇÃO

Em um mundo onde os recursos naturais estão cada vez mais limitados, a definição de políticas e práticas para se minimizar o impacto das ações humanas sobre o meio ambiente tem tido importância crescente. Representar os processos inerentes e relacionados ao meio ambiente através de modelos computacionais possibilita investigar o sistema como um todo, incorporando propriedades tais como resiliência e retroalimentação. Este tipo de abordagem possui aplicações nas mais diversas áreas da ciência, como hidrologia, mudança de uso e cobertura da terra, ecologia e sistemas sociais.

Não é fácil modelar computacionalmente as relações entre o homem e o meio ambiente. É preciso seguir vários passos. Primeiro, obter e armazenar os dados relativos à área de estudo. Depois, conceber e implementar o modelo. Finalmente, avaliar os resultados da simulação para verificar se o modelo nos ajuda a melhor compreender a realidade. Para responder a estes desafios, precisamos de ferramentas computacionais cujos conceitos sejam próximos da realidade do modelador. Tais ferramentas devem representar o espaço geográfico de forma explícita, usualmente na forma de partições regulares. Waldo Tobler (1979), dentro do campo da geografia, foi quem primeiro propôs a idéia de uma “Geografia Celular”.

Nos modelos tratados nesta tese (GARDNER, 1970; SCHELLING, 1971; DE KONING et al., 1999), as partições regulares do espaço são chamadas de espaços celulares. Nesta representação, o espaço é dividido em células de tamanho regular conectadas através de relações de vizinhança. Na literatura, o uso de espaços celulares é a forma mais usual de representação computacional do espaço geográfico em modelos sociedade-natureza.

Nos últimos anos, o volume de dados disponíveis para os pesquisadores realizarem simulações computacionais sobre as relações sociedade-natureza vem aumentando consideravelmente. Novas tecnologias têm possibilitado a aquisição de dados com resoluções cada vez mais finas e com maior

frequência temporal, armazenados nos mais diversos formatos (GRAY et al., 2005). Assim, para que as ferramentas de modelagem ambiental possam ser usadas amplamente pela comunidade científica, é necessário que sejam capazes de tratar grandes volumes de dados. Para que isto seja possível, há barreiras técnicas a vencer.

Na área de modelagem ambiental, a literatura apresenta um grande número de ferramentas para diversas aplicações. As principais ferramentas de simulação (CARNEIRO et al., 2012; NORTH et al., 2006; PCRASTER TEAM, 2012) possuem capacidade limitada de gerenciamento de dados, por usarem apenas a memória primária do computador. O encargo pelo gerenciamento de memória é dos desenvolvedores, que criam e implementam estruturas de dados em memória. Na maior parte dos casos, isto é um fator limitante nos programas escritos com estas ferramentas. A não ser que o ambiente computacional conte com grande quantidade de memória primária, os mecanismos de memória virtual disponíveis nos sistemas operacionais presentes em computadores de uso pessoal, não são apropriados para as aplicações de modelagem. Devido a estas limitações, muitos modeladores degradam a resolução espacial do modelo ou reduzem a área de estudo de forma a diminuir os custos computacionais, tornando o estudo viável.

Ao mesmo tempo, o campo de banco de dados, que tem produzido tecnologias para o armazenamento e gerenciamento de dados desde o final dos anos 60 (ELMASRI; NAVATHE, 2006), vive um momento de grande debate sobre a necessidade de repensar suas tecnologias, tanto do ponto de vista de suas arquiteturas e linguagens quanto de nichos mais específicos de aplicações (STONEBRAKER; CENTINTEMEL, 2005; STONEBRAKER et al., 2007; MADEN, 2012). As novas demandas produzidas pelos requisitos dos novos negócios em torno de dados e informações na Web por um lado, e as do campo científico, com dados espaço-temporais em profusão, têm estabelecido duas tendências neste debate. Uma perspectiva que chamaremos aqui de campo NoSQL (CATTELL, 2010) e outra apontando a necessidade de uma

abordagem que toma o passado construído e consolidado em teoria e tecnologias no campo de banco de dados e avança em função das necessidades apontadas pelas novas dinâmicas das aplicações no século XXI (STONEBRAKER et al., 2007; MADDEN, 2012).

Esta tese é construída nesta interface, com o foco nas ferramentas computacionais desenhadas para lidar com a modelagem de processos do domínio geográfico que, por terem sido projetadas para tratar com um volume de dados mais restrito e nascidas, muitas vezes, fora do campo da ciência da computação, não contaram com as tecnologias de banco de dados associadas ao seus desenvolvimentos. Portanto, o desafio que enfrentamos neste trabalho é o de gerenciamento de dados para esta classe específica de ferramentas de modelagem.

1.1. A Contribuição desta Tese

Considerando os cenários apresentados, esta tese procura contribuir para os estudos de modelagem ambiental integrada, associando às ferramentas computacionais de modelagem desta área uma maior capacidade de tratar com dados reais em grande volume. O desafio enfrentado é permitir o gerenciamento de um volume de dados considerável para estas ferramentas, com impacto mínimo do ponto de vista do modelador.

A hipótese defendida nesta tese é que as ferramentas de modelagem ambiental podem se beneficiar de alteranativas atuais de gerenciamento de dados através da concepção de uma arquitetura de software, materializada na forma de um *middleware* (NAUR; RANDELL, 1969), que atenda a seus requisitos de gerenciamento de dados. Assim sendo, esta tese propõe um *middleware* que gerencia o acesso e o armazenamento de *espaços celulares*. Ele permite que as ferramentas de modelagem possam ser desenvolvidas de forma independente das tecnologias de armazenamento de dados. A proposta resulta de uma análise dos requisitos de ferramentas de modelagem. Procuramos entender o que há de comum nas aplicações típicas e separar as

funções de processamento das funções de acesso e armazenamento. Os resultados obtidos a partir de um protótipo desenvolvido deste *middleware* demonstram a capacidade da arquitetura proposta em tratar o problema de modelar com grande volume de dados em ambiente de computador pessoal.

1.2. Estrutura deste Documento

Esta tese esta escrita da seguinte forma. No Capítulo 2 é realizada uma revisão da literatura sobre as ferramentas de modelagem dinâmica espacial, bem como das novas tecnologias de armazenamento. O Capítulo 3 apresenta a arquitetura proposta, materializada no middleware CellDB, e o Capítulo 4 descreve estudos sobre a capacidade desta ferramenta. Finalmente, o Capítulo 5 apresenta as conclusões e trabalhos futuros.

2 REVISÃO DA LITERATURA

No campo das tecnologias de banco de dados, desde meados dos anos 90, um grande debate científico internacional vem sendo realizado sobre as bases técnicas e conceituais dos sistemas de banco de dados e sua adequação às demandas originadas a partir das novas tecnologias de produção e coleta de dados em grande escala e das necessidades que toda uma nova geração de aplicações com uso intensivo da *web* trouxeram para o cenário atual. Trabalhos como o de Stonebraker e Cetintemel (2005) mostram que existe uma tendência de uma maior fragmentação das tecnologias de banco de dados em nichos mais específicos de aplicações. Este cenário é reforçado com o surgimento de novas arquiteturas de banco de dados e linguagens de manipulação de dados voltadas às necessidades das aplicações da comunidade científica (CUDRE-MAUROUX et al., 2009; KERSTEN et al., 2011; DOBOS et al., 2011).

Um campo técnico-científico que teve grande crescimento foi o da modelagem socioambiental. A existência de novos sistemas de observação e coleta de dados ambientais, climáticos e sóciodemográficos e as novas dimensões das questões relacionadas com mudanças nas relações população-ambiente, em um contexto de escala global, recolocaram os modelos e simulações computacionais de processos físicos e sociais na agenda científica e com impactos nos sistemas de decisão nas escalas local, regional e global.

Se por um lado já havia uma tradição de modelagem computacional intensiva no campo da meteorologia e do clima, novos problemas passaram a demandar uma abordagem de modelagem mais integrada do sistema terrestre, exigindo novos modelos e ferramentas computacionais que pudessem, além de tratar os processos físicos, lidar também com suas interações com os processos sociais estabelecidos em espaços geográficos concretos e complexos. Neste contexto nasceu ou renasceu um conjunto de ferramentas computacionais destinadas a tratar parcialmente elementos presentes no domínio dos novos problemas. Um subconjunto destas ferramentas buscou apresentar metodologias e técnicas

para modelar e simular dinâmicas presentes em espaços geográficos computacionalmente representados, utilizando, em princípio, dados e informações produzidas a partir de sistemas de informação geográfica e bancos de dados espaciais.

A relação entre este subconjunto de ferramentas, denominado ferramentas de modelagem dinâmica espacialmente explícitas, e as novas tecnologias de banco de dados é o objeto central desta tese. Este capítulo procura fazer uma revisão da literatura com base em trabalhos relacionados a este recorte temático, construindo as bases para uma proposta de arquitetura de um sistema de banco de dados que atenda às necessidades das ferramentas de modelagem dinâmicas espacialmente explícitas, desenhadas para uso como aplicativos monousuários executados em um computador pessoal.

2.1. Ferramentas de Modelagem de Dinâmicas Espacialmente Explícitas: Modelagem Dinâmica Espacial

As ferramentas de modelagem dinâmica espacial têm como principal objetivo facilitar a concepção, implementação e simulação de modelos, através da apresentação de conceitos que mapeiam abstrações do mundo real para representações computacionais. Estes conceitos abstraem as estruturas de dados internas da ferramenta, fazendo com que os modeladores possam focar na definição das entidades que compõem o modelo, bem como nos seus comportamentos e nas suas relações. Assim, o modelador não necessita se preocupar com aspectos computacionais, tais como sincronização de processos e entrada e saída de dados.

Atualmente, existe um conjunto de ferramentas capazes de trabalhar com a criação de modelos dinâmicos espaciais. Apresentamos aqui uma breve revisão de três destas ferramentas, baseadas em software livre, disponíveis atualmente: *TerraME* (CARNEIRO et al., 2012), *RePast* (NORTH et al., 2006) e *PCRaster* (PCRASTER TEAM, 2012). Nosso enfoque não é de apresentar uma revisão exaustiva de todas as funcionalidades destas ferramentas, nem a

comparação entre os conceitos introduzidos por cada uma delas, uma vez que existem muitos trabalhos com este fim (GILBERT; BANKES, 2002; TOBIAS; HOFMANN, 2004; RAILSBACK et al., 2006; ANDRADE, 2010). Nosso foco nesta seção são as estruturas utilizadas para representação de dados e seu acesso.

2.1.1. TerraME: Terra Modeling Environment

O *TerraME* é uma ferramenta de modelagem que possibilita a criação de modelos nos quais os componentes espacial, temporal e comportamental podem ser especificados de forma independente unss dos outros (CARNEIRO et al., 2012). Distribuído como software livre, sob a licença GNU LGPL (http://www.terrame.org), o ambiente fornecido pelo TerraME possibilita a escrita de modelos através da linguagem de programação Lua (IERUSALIMSCHY et al., 1996). A sua estrutura geral é mostrada na Figura 2.1.

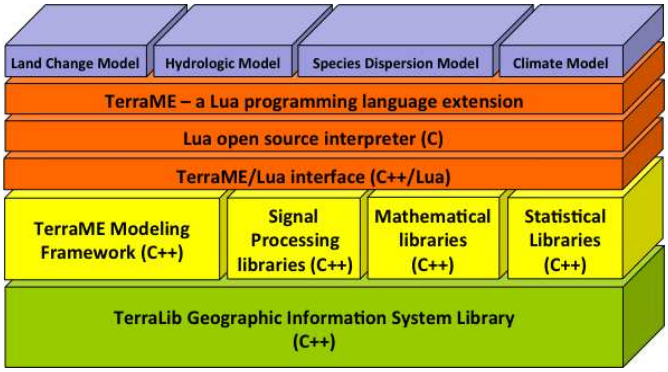


Figura 2.1 - Estrutura Geral do TerraME.
Fonte: Carneiro et al. (2012)

No nível mais baixo dessa arquitetura, o TerraME utiliza os serviços da biblioteca TerraLib (CÂMARA et al., 2010) para armazenamento e recuperação dos dados em bancos de dados geográficos. As camadas de informação da TerraLib, com representações de pontos, linhas, polígonos ou células, são usadas para carregar os atributos das células de uma estrutura denominada

espaço celular (CARNEIRO et al., 2008). Esta estrutura faz parte do sistema de tipos espaciais do TerraME. Os atributos das células são todos armazenados em memória primária, numa estrutura de dados híbrida. Os atributos manipulados nos modelos são representados por campos de tabelas da linguagem Lua, enquanto alguns atributos de controle são representados por uma estrutura de dados implementada na linguagem C++ (STROUSTRUP, 1997). Esta estratégia possibilita uma máxima flexibilidade em relação aos tipos de dados que podem ser utilizados para representar os atributos das células, uma vez que a linguagem Lua é fracamente tipada.

Outra estrutura de dados amplamente utilizada nos modelos escritos com o TerraME é denominada *vizinhança*. A *vizinhança* pode ser carregada a partir de um banco de dados TerraLib contendo uma *Matriz de Vizinhança Generalizada* (*GPM*) (AGUIAR, 2006), lida de um arquivo, ou ainda computada dinamicamente. Nos dois casos os relacionamentos entre as células são materializados na memória. Cada célula carrega uma ou mais listas de adjacência, dependendo do número de vizinhanças associadas a ela. Além dos tipos mencionados acima, o TerraME ainda possui tipos comportamentais (Agente e Autômato) e temporais (Evento e *Timer*).

2.1.2. RePast: Recursive Porous Agent Simulation Toolkit

O *RePast* é uma ferramenta de modelagem com base no conceito de agentes (NORTH et al., 2006). Ela possui suporte às linguagens Java (GOSLING et al., 2005), Python (VAN ROSSUM, 1993) e para a plataforma .Net da Microsoft (MICROSOFT, 2012). Os agentes nesta ferramenta não possuem nenhum tipo de representação espacial explícita. Para possibilitar a criação de relacionamentos entre os agentes assim como para inseri-los em um espaço, existe o conceito de contexto (*Context*). Um contexto agrupa os agentes em um *container* onde podem ser aplicadas uma ou mais projeções (*Projection*). As projeções possibilitam a criação de estruturas para definição dos espaços onde

os agentes serão inseridos. Conforme descrito por CROOKS (2007), o RePast suporta três tipos de espaços:

a) *espaços celulares*: representados entre outras formas por grades regulares ou hexagonais. As células dessas grades podem conter um único agente ou múltiplos agentes, dependendo das restrições usadas na sua construção. Dados geográficos matriciais, no formato *ESRI ASCII*, podem ser usados na importação e exportação de dados nesse tipo de representação do espaço. A única limitação é que cada espaço pode conter uma única propriedade. A representação de grade possibilita a realização de consultas de vizinhanças como a de Moore (vizinhança 8) e Von Neumann (vizinhança 4);

b) *espaços contínuos*: a localização dos agentes pode ser representada por um par de coordenadas em ponto flutuante ou pela associação a uma determinada geometria vetorial (pontos, linhas e polígonos), no caso de espaços contínuos geográficos. Os agentes neste último caso podem ter suas informações carregadas a partir de arquivos *shapefile*, onde cada feição contida no arquivo dá origem a um agente do modelo. Este suporte geográfico é realizado através das bibliotecas *GeoTools* (TURTON, 2010) e *Java Topology Suite* (VIVID SOLUTIONS, 2012). Consultas espaciais como a busca por retângulos ou que envolvam operações topológicas entre as geometrias associadas aos agentes podem ser realizadas sobre o espaço geográfico;

c) *espaço de redes*: o conceito de redes do RePast possibilita a criação de relacionamentos entre os agentes. Esses relacionamentos podem ser criados de forma dinâmica ou carregados a partir de dados armazenados em algum formato textual. Diferentemente do TerraME, no RePast a estrutura de rede é um *container* (*Network*) do mesmo nível que o espaço celular ou que o espaço contínuo.

Assim como no TerraME, todos os dados manipulados pelo RePast, *agentes*, *geometrias*, *células* e *vizinhança*, são armazenados diretamente na memória primária durante a execução das simulações.

2.1.3. PCRaster: Software for Environmental Modeling

O *PCRaster* é uma ferramenta de modelagem baseada em uma álgebra de mapas tradicional do universo SIG (PCRASTER TEAM, 2012). Os dados dos modelos são discretizados na forma de células. Cada uma delas pode conter um ou mais atributos, mas apenas um será usado para receber ou transmitir informações para as células vizinhas.

Em relação ao armazenamento de dados, o PCRaster possui um formato matricial próprio. Cada espaço de atributos é armazenado em um arquivo separado (chamado de mapa). Uma pilha de mapas é usado para compor o ambiente sendo modelado. A propriedade de uma célula é formada pelos valores dos atributos em cada mapa (Figura 2.2).

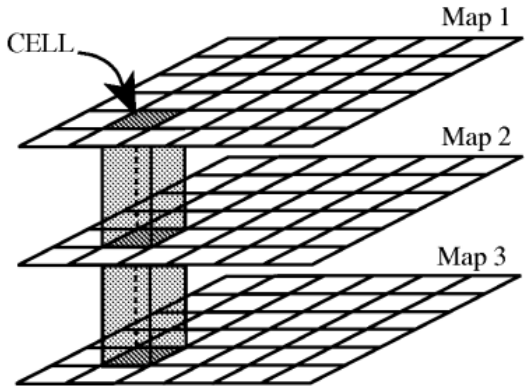


Figura 2.2 - O ambiente de um modelo representado como um conjunto de mapas.
Fonte: PCRaster Team (2012)

As vizinhanças são definidas apenas considerando-se as adjacências das células ou através de mapas com caminhos definidos entre as células. Portanto, não há flexibilidade, como no *RePast* ou *TerraME*, para este tipo de

dados. Em relação ao intercâmbio de dados, é preciso realizar manualmente a conversão deste formato interno para ASCII e vice-versa.

2.1.4. Considerações Sobre as Ferramentas de Modelagem

Como mencionado no início da Seção 2.1, atualmente existem diversas ferramentas de modelagem. A Figura 2.3, extraída de Andrade (2010), apresenta outra lista de ferramentas, onde é apresentada uma comparação de como cada uma delas lida com conceitos de dados espaciais. Assim como nas ferramentas apresentadas anteriormente, todas elas manipulam de alguma forma dados relacionados aos conceitos de *Espaços Celulares* e *Vizinhanças*. Com exceção do *PCRaster*, todas as demais realizam a carga de dados diretamente para a memória primária do computador. Assim, nenhuma delas é capaz de trabalhar bem com esses conceitos quando o volume de dados ultrapassa a quantidade de memória livre do hardware usado na simulação.

Toolkit	**Espaço**	**Agente**	**Vizinhança**		**Localização**	
			Agente	**Célula**	**Agente → Célula**	**Célula → Agente**
Swarm	Matricial	[2]	[2]	Euclidiano	Manipula	[2]
Netlogo	Matricial	[2]	[2]	Euclidiano	Manipula	[2]
Repast	Matricial, Vetorial	Vetorial	[2]	Complexo	Manipula, Cria	[2]
OBEUS	Vetorial	Vetorial	Transição	Complexo	Manipula, Cria	Manipula, Cria

[2] A cargo do modelador

Figura 2.3 - Comparação entre diversas ferramentas de modelagem em relação ao trabalho com dados espaciais.

Fonte: Adaptado de Andrade (2010)

Outra questão que fica evidente nessas ferramentas é a falta de capacidade em lidar com uma maior variedade de formatos para entrada e saída de dados espaciais nos modelos. Por serem concebidas para grupos de pesquisa e usuários com diferentes perfis, a questão de acesso aos dados é importante para ampliar e facilitar o seu uso. Esta questão torna-se ainda mais relevante quando se considera que, apesar de todos os esforços realizados pela indústria

geoespacial nos últimos anos para padronização de formatos e serviços de intercâmbio de dados geográficos (CASANOVA et al., 2005), a cada dia surgem novos formatos e sistemas.

2.2. Novas tecnologias de Gerenciamento de Bancos de Dados

Nos últimos anos, a busca por gerenciadores de bancos de dados mais eficientes e especializados em certos nichos de aplicações tem estimulado o desenvolvimento de novas tecnologias com características diferentes dos *Sistemas de Gerenciamento de Bancos de Dados Relacionais* (SGBD-R) (ELMASRI; NAVATHE, 2006). Sistemas como o *Neo4J* (HUNGER, 2010), *Apache CouchDB* (ANDERSON et al., 2010), *MongoDB* (CHODOROW; DIROLF, 2010), *Apache Cassandra* (LAKSHMAN; MALIK, 2010) e *SciDB* (BROWN, 2010) fornecem modelos de dados, linguagens de consulta e interfaces de programação específicos.

Esses novos sistemas estão sendo denominados de *NoSQL*, acrônimo de *Not Only SQL*, que embora não seja um termo de total consenso na comunidade, tem sido usado para enfatizar a diferença com os SGBD-R. Para entender melhor este debate, apresentamos uma revisão da literatura de banco de dados partindo dos SGBD-R.

2.2.1. SGBDs Relacionais

Os SGBD-R têm sido empregados para o armazenamento e gerenciamento de dados nos mais diversos tipos de aplicações. Na última década, estes sistemas se tornaram capazes de lidar de maneira ampla e eficiente com dados geográficos através da inclusão de tipos de dados, operadores e mecanismos de indexação espaciais (CASANOVA et al., 2005). O setor comercial desenvolveu o Oracle Spatial (ORACLE, 2003), o IBM DB2 Spatial Extender (IBM, 2002) e, mais recentemente, a Microsoft introduziu suporte espacial ao SQL Server (MICROSOFT, 2008). A comunidade de software livre também

criou extensões, como o PostGIS para o PostgreSQL (OBE et al., 2011) e o SpatiaLite para o SQLite (FURIERI, 2012).

Grande parte do suporte espacial dos atuais SGBD-R é baseado em padrões abertos, como a Simple Feature Specification for SQL (OGC, 2012a; OGC, 2012b) e a SQL/MM (MELTON e EISENBERG, 2001). Estas especificações abrangem basicamente o modelo geométrico vetorial e as operações espaciais que devem ser suportadas pelas implementações. O modelo geométrico contém representações para pontos, linhas, polígonos, coleções homogêneas e heterogêneas de geometrias, como mostrado na Figura 2.4. Os operadores topológicos seguem o paradigma da matriz de 9-interseções estendida dimensionalmente, proposta por Clementini e Di Felice (1995).

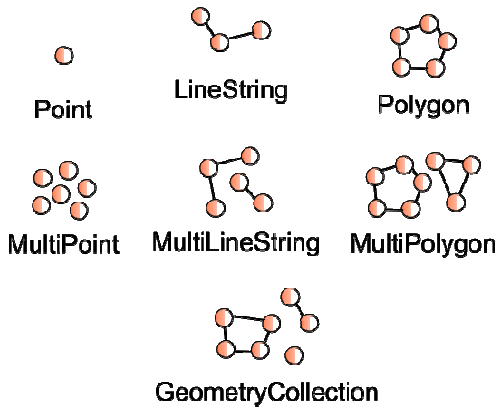


Figura 2.4 - Tipos Geométricos Básicos da SFS-SQL.

A introdução dos tipos geométricos nos SGBD-R tornou possível a criação de tabelas em que uma das colunas é capaz de armazenar geometrias. O exemplo mostrado na Figura 2.5 usou coleções homogêneas de polígonos para representar as áreas de cada unidade federativa. Cada objeto armazenado desta forma possui todos os pares de coordenadas da fronteira. Consequentemente, os relacionamentos espaciais são computados durante a execução das consultas, não sendo armazenados de forma explícita como em modelos topológicos.

Tabela: unidades_federativas		
gid	sigla	geometria
1	MG	
2	SP	
3	RJ	

Figura 2.5 - Exemplo de tabela com coluna geométrica.

A inclusão de operadores espaciais possibilitou a criação de consultas SQL como a mostrada na Figura 2.6, em que a região com a aptidão agrícola do município de João Pinheiro no Estado de Minas Gerais é calculada a partir de dados armazenados em duas tabelas, uma com o mapa de municípios do Estado de Minas Gerais e outra com o de aptidão agrícola deste Estado.

```
SELECT m.nommuni, a.classe,
       ST_Intersection(m.the_geom,
                       a.the_geom)
  FROM mg_municipios m,
       mg_aptidao_agricola a
 WHERE ST_Intersects(m.the_geom,
                     a.the_geom)
   AND m.nommuni = 'João Pinheiro'
```

Figura 2.6 - Consulta espacial em SQL.

O PostGIS, o Oracle Spatial e o SpatiaLite optaram por fornecer mecanismos de indexação espacial baseados nas *Árvores-R* (GUTTMAN, 1984). Outros, como o SQL Server e o IBM DB2 Spatial Extender, utilizam uma estratégia baseada em *Grades Fixas Multiníveis* (NIEVERGELT et al., 1984) e *Space Filling Curves* (LAWDER e KING, 2000), de forma a não terem a necessidade

de criar novas implementações de seus mecanismos de indexação subjacentes, as *Árvores-B*$^{+}$ (COMER, 1979).

No entanto, cada vez mais, novas funcionalidades geoespaciais são integradas aos SGBD-R. Uma das mais recentes é o armazenamento e gerenciamento de dados na forma matricial (*raster*). Como exemplo, a extensão PostGIS Raster (OBE et al., 2011) lançada oficialmente em 2012, possibilita armazenar no banco de dados imagens georreferenciadas. Esta extensão disponibiliza diversas estratégias de armazenamento da imagem, entre elas: (a) divisão da imagem em pequenos blocos, denominados *tiles*, que são armazenados em tabelas (Figura 2.7), e (b) armazenamento de referências (links) para as imagens gravadas em arquivos fora do banco. Também, é possível criar pirâmides (ou *overviews*) para acelerar a visualização por parte das aplicações. O conjunto de operadores e funções presentes nessa extensão permite a realização de análises complexas nas imagens, assim como operações envolvendo dados vetoriais e matriciais, como o recorte de uma imagem a partir de uma máscara vetorial.

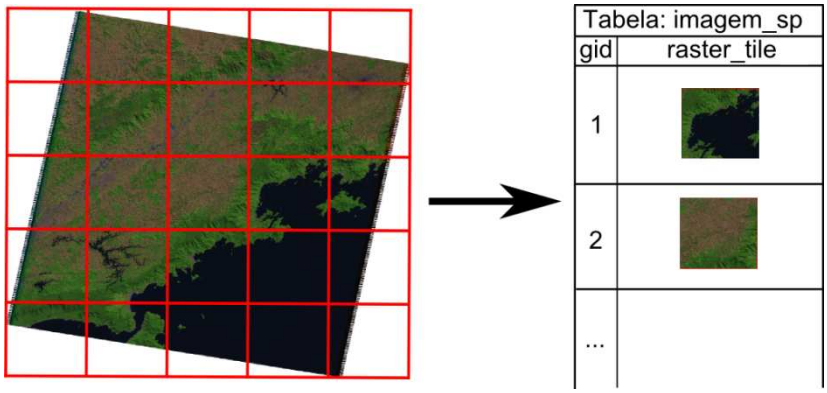


Figura 2.7 - Imagem armazenada em uma tabela.

Outras funcionalidades que vêm sendo incluídas nos SGBD-R são voltadas para representações topológicas. Extensões como o PostGIS Topology introduzem tipos e operadores para trabalhar com os conceitos de nós, arestas e faces. Neste tipo de representação, os pontos das fronteiras dos objetos são

armazenados uma única vez, sendo compartilhados entre as faces. Este suporte pode ser usado, entre outras coisas, para garantir a correta topologia das áreas geradas como resultado de processos de simplificação.

Por último, existem extensões dos SGBD-R para lidar com representações de rede. Apesar da existência da extensão pgRouting para o PostgreSQL, apenas o Oracle Spatial Network Data Model (ORACLE, 2008) possui um modelo mais amplo e com operadores capazes de atender a um maior domínio de aplicações geoespaciais.

Do exposto até aqui, observa-se que a tecnologia relacional integrou várias funcionalidades que antes eram encontradas apenas nos Sistemas de Informação Geográficas (SIG). Em consequência disso, os SGBD-R têm se mostrado uma solução eficiente para o gerenciamento de dados geoespaciais. Já a padronização desse suporte teve o efeito positivo de possibilitar o desenvolvimento de uma geração de aplicativos geográficos interoperáveis e capazes de utilizar os SGBD-R.

Atualmente, as principais bibliotecas de acesso a dados geoespaciais baseadas em software livre, GDAL (WARMERDAM, 2010), GeoTools (TURTON, 2010) e TerraLib (CÂMARA et al., 2010), possuem uma boa interface com sistemas relacionais. Aplicativos desktop, como o Quantum GIS (QUANTUM GIS DEVELOPMENT TEAM, 2012) e TerraView (CÂMARA et al., 2010), também são capazes de visualizar e analisar dados destes sistemas. O GeoServer (AIME, 2007), MapServer (KROPLA, 2005) e TerraOGC (CÂMARA et al., 2010) disponibilizam serviços web geográficos a partir de bases de dados relacionais.

2.2.2. Bancos de Dados Orientados a Documentos

Os sistemas desta classe de tecnologia, também denominada de *Document Stores*, utilizam a notação *JSON - JavaScript Object Notation* (CROCKFORD, 2006), para representação dos dados. JSON é um formato leve para

intercâmbio de dados baseada na linguagem *JavaScript* (ECMA INTERNATIONAL, 2011) que popularizou-se entre os desenvolvedores de aplicações Web. Os documentos não possuem obrigatoriamente um esquema global compartilhado, o que possibilita definir dinamicamente novos atributos sem a necessidade de alteração de outros documentos previamente armazenados. Atualmente, o sistema baseado em software livre mais popular desta família de tecnologias é o *MongoDB* (CHODOROW; DIROLF, 2010).

O MongoDB é escrito em C++ sob a licença GPL. Um banco de dados neste sistema é composto por coleções de documentos no formato BSON, uma versão binária dos documentos JSON. Cada documento possui um atributo identificador obrigatório ("_id") usado para manter um índice baseado em uma Árvore-B. O seu mecanismo de consulta permite utilizar expressões JSON para expressar filtros sobre os atributos dos documentos. Como nos sistemas relacionais, os índices são escolhidos automaticamente pelo servidor durante o processamento das consultas. O suporte geoespacial do MongoDB permite a criação de índices espaciais sobre dados representados por pontos 2D através de uma técnica conhecida como *Geohash (10GEN, 2013)*. Este tipo de índice pode ser explorado em consultas de localização, como os N vizinhos mais próximos a um determinado ponto e consultas por intervalo, utilizando-se um retângulo de busca, um círculo ou polígono simples como filtro.

Os primeiros sistemas geoespaciais construídos sobre essas tecnologias lidam basicamente com o armazenamento de pontos de interesse (POI) e tiles de imagens, comuns em aplicações de mapas na Web ao estilo do Google Maps. No caso específico do MongoDB, alguns trabalhos estão começando a explorá-lo para o armazenamento e processamento de dados matriciais (WADSWORTH, 2012); enquanto outros estão adicionando novos índices espaciais baseados em Árvores-R para dar suporte ao trabalho com tipos geométricos mais complexos (KNIZE, 2012).

A capacidade de lidar com dados sem um esquema rígido, combinada com eficientes sistemas de indexação, torna esses sistemas uma solução interessante para a construção de aplicativos da Web 2.0.

2.2.3. Armazenamento baseado em pares chave-valor

Também conhecidos por *Key-Value Stores*, os sistemas derivados do DBM (GNU, 2011), como Oracle Berkeley DB (OLSON et al., 1999), Kyoto Cabinet (FAL LABS, 2011) e LevelDB (DEAN; GHEMAWAT, 2012), são bibliotecas com rotinas para o gerenciamento de bancos de dados baseadas no armazenamento de pares chave-valor. A chave é usada para identificar um valor que pode ser um dado estruturado ou não. Estes sistemas oferecem vários métodos de acesso, como Árvores-B^+ e Hash, que são utilizados na organização primária dos registros do banco de dados, além de componentes configuráveis, como cache de dados, controle de concorrência, mecanismos de transação, recuperação a falhas e até mesmo replicação. Em geral, podem ser embutidos nas aplicações, de forma a evitar gargalos de comunicação entre processos comuns nas arquiteturas cliente-servidor dos sistemas relacionais. Este modelo de biblioteca fornece um nível de programação apropriado para a criação de soluções especializadas de armazenamento. Um exemplo disso é o caso de modelos de dados complexos que não se enquadram bem no modelo relacional, como grafos e documentos não estruturados.

Tradicionalmente, esses sistemas têm sido amplamente utilizados de forma embarcada em dispositivos móveis (celulares) e roteadores de rede, em sistemas de autenticação de usuários e de gerenciamento de conteúdo. Outro exemplo de uso pode ser encontrado em Decandia et al. (2007), que descreve o sistema Dynamo, desenvolvido pela Amazon com o objetivo de suportar as demandas exigidas na manutenção de estado das suas aplicações: alta disponibilidade e escalabilidade.

2.2.4. Bancos baseados em Grafos

Estes sistemas se baseiam na Teoria dos Grafos (BOLLOBÁS, 1998) e são conhecidos como *Graph Databases*. Fornecem três construtores básicos em seu modelo de dados: nós (ou vértices), ligações (ou arestas) e propriedades. Os nós são usados para modelar objetos que existem de forma independente das ligações. Em geral, não possuem um esquema rígido. Assim, dois objetos representados por nós de um mesmo grafo podem ter atributos bem distintos. As arestas são usadas para materializar o relacionamento entre nós. As propriedades podem ser usadas tanto para descrever atributos de nós quanto de arestas.

Este modelo de dados tem se tornado popular em aplicações web de domínio social, fornecendo capacidades interessantes para a realização de consultas requeridas por este tipo de aplicação e com um bom desempenho. As implementações desta tecnologia, como o *OrientDB* e o *Neo4J* (HUNGER, 2010), optaram por integrar uma linguagem para travessia de grafos denominada de *Gremlin* (AVRAM, 2010), capaz de expressar travessias complexas nos grafos de forma concisa, ao invés do uso estrito da SQL.

O Neo4J possui uma extensão chamada *Neo4J Spatial* (NEUBAUER, 2011) que acrescenta métodos de indexação espacial e operadores espaciais a este sistema. Esta biblioteca funciona integrada ao GeoTools, possibilitando o acesso a dados armazenados no Neo4J por aplicações como o GeoServer e uDIG.

2.2.5. Bancos de Dados Matriciais

Como alertado por Jim Gray e colegas (GRAY et al., 2005), sempre houve uma lacuna entre as tecnologias de banco de dados relacionais e a comunidade científica. Entre os vários pontos deste descasamento, o modelo de dados e a linguagem de consulta se mostraram grandes barreiras para adoção dessa

tecnologia, pois os dados científicos geralmente se encontram na forma de matrizes multidimensionais.

Em consequência, vários grupos optaram por construir seus próprios sistemas de armazenamento e processamento de dados. Um exemplo desta prática é mostrado em Raoult (2012), que apresenta a arquitetura do *MARS*, um sistema desenvolvido pelo Centro Europeu de Previsão de Médio Prazo (ECMWF) para lidar com mais de 30.000 requisições diárias, acessando cerca de 1.500.000 imagens, que correspondem a mais de 100 GB de dados recuperados para processamento todos os dias.

No entanto, mais recentemente, um consórcio formado por diversos pesquisadores da área de banco de dados e representantes das comunidades científicas de Astronomia e Sensoriamento Remoto criaram um sistema chamado *SciDB* (CUDRE-MAUROUX et al., 2009; BROWN, 2010). Voltado para as demandas dessas comunidades, este sistema fornece um modelo de dados e uma linguagem de consulta baseados em matrizes multidimensionais (Figura 2.8). Esse modelo tem dado origem à nomenclatura de *Array Databases*.

```
create array observacoes
<sensor:string, temperatura:float>
[I=0:4, J=0:4];
```

J \ I	[0]	[1]	[2]	[3]
[0]	("A", 29)	("K", 32)	("B", 31)	("C", 32)
[1]	("F", 30)	("D", 31)	("E", 30)	("U", 32)
[2]	("O", 32)	("V", 32)	("L", 32)	("X", 32)
[3]	("R", 33)	("S", 33)	("Q", 32)	("T", 32)

Figura 2.8 - Definição de uma matriz 4 x 4 com 2 atributos.
Fonte: Adaptado de Brown (2010)

O SciDB usa uma estratégia de armazenamento onde a matriz é quebrada em blocos que podem ser distribuídos em diversas máquinas para acelerar a realização de análises em matrizes muito grandes. Além disso, os blocos podem replicar partes das bordas para favorecer a computação de vizinhança. Estratégias de compactação com algoritmos que fazem um balanceamento entre capacidade de compressão e uso de CPU podem ser aplicadas a esses blocos. Os valores de cada célula são armazenados em arquivos separados, pois é comum, nas aplicações científicas, ter-se matrizes com células com grande número de valores e modelos, e processos que usam apenas alguns deles. Essa separação também tem como objetivo obter um melhor resultado de compressão.

2.3. As Ferramentas de Modelagem Dinâmica Espacial e as Novas Tecnologias de Banco de Dados

As novas tecnologias de armazenamento, gerenciamento e processamento de grandes volumes de dados têm sido criadas, basicamente, por dois grupos distintos: a dos gigantes da internet, como Google, Amazon e Facebook, e a dos grupos de pesquisa em bancos de dados associados a comunidades científicas, principalmente, de Astronomia e Sensoriamento Remoto. O primeiro grupo tem se preocupado com a criação de sistemas para atender aplicações Web que lidam com grande número de usuários, como máquinas de busca, redes sociais e demais aplicações da Web 2.0. Nestas aplicações, existe a necessidade de realizar análises sobre petabytes de dados provenientes de fontes não estruturadas, como documentos HTML, PDF, arquivos texto, mensagens eletrônicas, entre outras. Essas tecnologias surgiram, em grande parte, motivadas pelos trabalhos publicados pelo Google sobre sua API de programação distribuída MapReduce e de seu sistema de armazenamento Google File System (GHEMAWAT et al., 2003; DEAN; GHEMAWAT, 2008;

CHANG et al., 2008). Em geral, esses sistemas encontram-se em operação em grandes centros de dados.

O segundo grupo tem buscado criar sistemas mais adequados à comunidade científica, introduzindo novos modelos de dados e linguagens de consultas baseados em matrizes (*arrays*). Trata-se de uma tentativa de resposta à crescente demanda deste meio, que, com os avanços nos equipamentos de coletas de dados, como telescópios, sensores a bordo de satélites, Geo-Sensores e GPS, tem enfrentado dificuldades para o armazenamento, processamento e análise de quantidades massivas de dados. Algumas dessas tecnologias, como o SciDB, já começam a ser projetadas para possibilitar a realização de simulações sobre grandes volumes de dados.

No entanto, não existe uma única tecnologia que seja totalmente projetada tendo em vista as necessidades das ferramentas de modelagem apresentadas neste capítulo, que trabalham com estruturas conhecidas como *espaço celular*, para representar o espaço geográfico através de um conjunto de células conectadas por relações de *vizinhança*. Pelo lado das tecnologias de banco de dados, observa-se que os sistemas baseados em grafos possuem um modelo de dados capaz de representar os espaços celulares. Cada célula pode ser mapeada para um objeto do tipo nó e ter conexões com as células de sua adjacência. No entanto, o espaço de armazenamento necessário por uma representação como esta se mostra inviável na prática se o espaço celular for de grandes dimensões.

Os bancos de dados matriciais possuem um modelo de dados e arquitetura mais adequados a esse tipo de representação. As matrizes multidimensionais são capazes de suportar a criação de espaços celulares com vários atributos, incluindo valores numéricos, textuais e datas. O armazenamento especializado com blocagem, compressão e replicação das bordas, possibilita explorar a travessia das células e da vizinhança espacial das células de forma eficiente. A distribuição de partes da matriz por várias máquinas torna possível a obtenção

de um alto grau de paralelismo nas operações de leitura e escrita das células. Os primeiros sistemas desta classe ainda são protótipos experimentais que funcionam em *cluster* de computadores, com um alto grau de dificuldade de instalação, configuração e uso. O suporte espacial é limitado, não possuindo tratamento de sistemas de referência espacial e operadores espaciais. A capacidade de importação e exportação de dados também é outro ponto que deixam a desejar.

Os SGBD-R com extensões matriciais também são adequados à representação dos espaços celulares. Embora a arquitetura destes sistemas não tenha o mesmo nível de sofisticação de bancos matriciais como SciDB, o suporte espacial delas é completo, incluindo várias ferramentas de conversão de dados. Em especial, a API de álgebra de mapas do PostGIS Raster possui elementos que facilitam alguns tipos de operações que são comuns nas ferramentas de modelagem: iteração pelas células e aplicação de funções que consideram a vizinhança espacial das células.

Voltando às ferramentas de modelagem, observamos também que nenhuma delas possui uma integração com as tecnologias de banco de dados de forma a suprir a falta de capacidade de manipulação e gerenciamento de grandes volumes de dados. Ainda temos que considerar que o principal ambiente fornecido por elas é voltado para computadores pessoais, portanto, equipamentos de pequeno porte. E todas possuem uma baixa capacidade de lidar diretamente com uma maior variedade de formatos para entrada e saída de dados espaciais.

Diante das novas demandas trazidas às ferramentas de modelagem por um tratamento mais eficiente de volumes de dados que podem ultrapassar facilmente a capacidade de manipulação em memória primária em computadores pessoais, e da necessidade de lidar com fontes de dados das mais diversas origens, para recuperação e armazenamento dos resultados finais e parciais das simulações, é preciso pensar em formas alternativas de

atender a estas demandas. A primeira parte dessa demanda poderia ser atendida através de uma melhor integração com as tecnologias de banco de dados discutidas acima, no entanto isto acarretaria uma forma muito diferente de operação das ferramentas de modelagem, que passariam a necessitar da instalação e configuração de sistemas complexos como os SGBD. A segunda parte da demanda só pode ser adequadamente atendida se a ferramenta tiver uma maior integração com bibliotecas geoespaciais como a TerraLib, GDAL e GeoTools. Mesmo utilizando essas bibliotecas, cada ferramenta terá que realizar um grande esforço para desenvolver estruturas de dados e formas de organização dos dados manipulados por elas.

Considerando as questões discutidas acima, uma possível solução para os problemas apresentados é a criação de um *middleware (NAUR; RANDELL, 1969) de integração de ferramentas de modelagem e sistemas de* armazenamento e gerenciamento de dados. A existência desse *middleware* viabilizaria trabalhar tanto com soluções em computadores pessoais quanto integradas a sistemas mais complexos. Neste sentido, como veremos no próximo capítulo, esta tese apresenta uma proposta de arquitetura para um *middleware* de bancos de dados voltado exclusivamente às necessidades deste nicho específico de aplicações de modelagem.

3 CellDB - Cellular DataBase: ARQUITETURA PARA ACESSO A DADOS NA MODELAGEM E SIMULAÇÃO AMBIENTAL COM USO DE ESPAÇOS CELULARES

Este Capítulo apresenta uma proposta de arquitetura para acesso eficiente a dados em ambientes de modelagem e simulação ambiental baseados em espaços celulares. Esta arquitetura estabelece um *middleware* de bancos de dados denominado ***CellDB*** - ***Cellular Database*** - que fornece uma camada de acesso e gerenciamento de grandes bancos de dados representados em espaços celulares. O CellDB desacopla as ferramentas de modelagem e simulação de dinâmicas espacialmente explícitas, no domínio dos problemas socioambientais, dos serviços necessários ao gerenciamento da massa de dados. Esta estratégia de desacoplamento produz para as ferramentas de modelagem e simulação baseadas em espaços celulares:

a) Capacidade para lidar com grandes volumes de dados. Passa a ser responsabilidade do CellDB fornecer, de forma transparente, mecanismos que possibilitem trabalhar com volumes de dados muito maiores do que a capacidade da memória primária, hoje uma limitação das ferramentas de modelagem e simulação em domínios espaciais no estado da arte;

b) Independência dos sistemas gerenciadores de dados. Através de uma arquitetura extensível, o CellDB permite o uso de novas tecnologias de armazenamento e recuperação de dados, sem a necessidade de alteração no código do subsistema de acesso a dados da ferramenta de modelagem utilizada, ampliando a capacidade de acompanhar a evolução das tecnologias de banco de dados;

c) Redução da complexidade do subsistema de acesso e gerenciamento de dados celulares. Com uma API (*Application Programming Interface*) projetada para lidar com as especificidades de acesso a dados que são representados em espaços celulares pelas ferramentas de modelagem,

o esforço de implementação dessas ferramentas pelos desenvolvedores é substancialmente reduzido, uma vez que os detalhes relativos aos diversos e diferentes protocolos e sistemas de armazenamento subjacentes são encapsulados pela API.

3.1. Concepção Geral do CellDB

O papel de *middleware* desempenhado pelo CellDB é mostrado na Figura 3.1. Além de desacoplar os ambientes de modelagem dos sistemas gerenciadores de dados (denominados deste ponto em diante de fontes de dados), o CellDB também pode ser usado como elo de ligação com os sistemas de informação geográficas (SIG). Dada a natureza dos dados manipulados pelas ferramentas de modelagem, o ambiente dos SIGs pode ser usado na preparação dos dados que serão utilizados nos modelos. A API do CellDB facilita este trabalho conjunto.

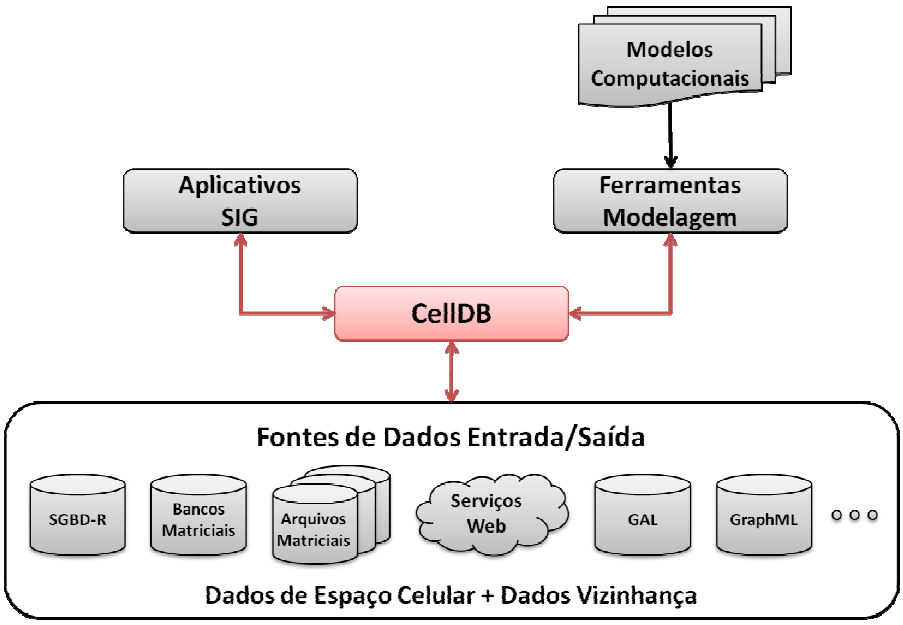


Figura 3.1 - *Middleware* CellDB.

Conceitualmente, o CellDB é formado por dois módulos (Figura 3.2):

1. *Camada de Acesso a Dados,* formada por um conjunto de abstrações para manipulação dos dados de espaços celulares e vizinhanças. Esta camada fornece operações para recuperação dos dados a partir de uma fonte de origem e para o direcionamento dos resultados intermediários gerados durante a execução das simulações para outras fontes. Ela também é responsável por manter, quando necessário, um armazenamento temporário para as simulações;

2. *Catálogo de Metadados* interno, com a definição de cada espaço celular e vizinhança a ser usado nas simulações.

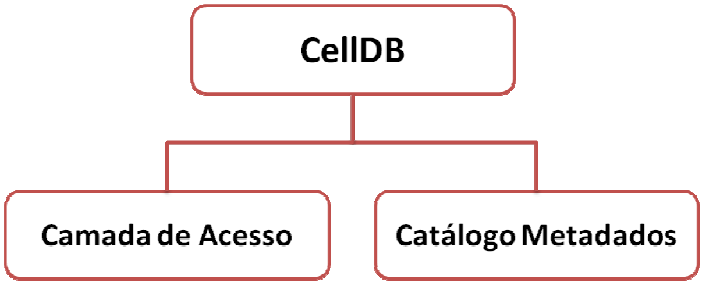


Figura 3.2 - Módulos do CellDB.

Os módulos *Camada de Acesso a Dados* e *Catálogo de Metadados* operam sobre uma estrutura de armazenamento conhecida como *espaço celular*. No Capítulo 2 os conceitos de *espaço celular* e *vizinhança* foram explorados como base de várias das ferramentas de modelagem de dinâmicas espaciais. Com base naquela análise e em recente trabalho de Carneiro et al. (2012), antes de detalhar os dois módulos do CellDB, é necessário descrever e caracterizar os tipos de dados envolvidos em espaços celulares que lidam com a manipulação de dados, essenciais a ferramentas de modelagem e simulação de dinâmicas espacialmente explícitas. Estes tipos de dados são descritos na próxima seção.

3.2. Tipos Abstratos de Dados em CellDB

Esta seção apresenta uma notação modificada para Tipos Abstratos de Dados (TAD) baseada em Guttag et al. (1978) para descrever os conceitos que fazem

parte da camada de abstração de dados do CellDB: ***célula, espaço celular e vizinhança***.

3.2.1. TAD Cell: Célula

Uma célula representa um elemento de um conjunto de elementos cuja união forma uma partição do espaço a ser representado. Uma célula tem uma geometria definida e um conjunto de atributos associados a ela. De acordo com seu uso nos modelos, as células podem ter atributos de três naturezas: estáticos, dinâmicos e sincronizáveis. Os atributos estáticos são usados apenas para leitura. Os atributos dinâmicos são usados tanto para leitura quanto escrita de valores. Os atributos sincronizáveis possuem uma cópia com a noção de estado passado e estado presente. Neste último tipo, os valores podem ser lidos do estado passado e escritos apenas no estado presente. A Figura 3.3 apresenta uma definição para o tipo abstrato de dado *Cell*, uma célula.

```
type  Attribute
syntax
   nature (Attribute) → Static | Dynamic | Synchronizable

type  Value: String, Int, Double, Bool

type  Cell uses Static, Dynamic, Synchronizable, Value
syntax
   new ([ (Attribute, Value) ]) → Cell
   read (Cell, Attribute) → Value
   current (Cell, Attribute) → Value
   write (Cell, Attribute, Value) → Cell
   synchronize (Cell, Attribute) → Cell
axioms
   c: Cell;  AV: [(Attribute, Value)], a: Attribute, v: Value

   c = new (AV) ⇒ AV ≠ ∅
```

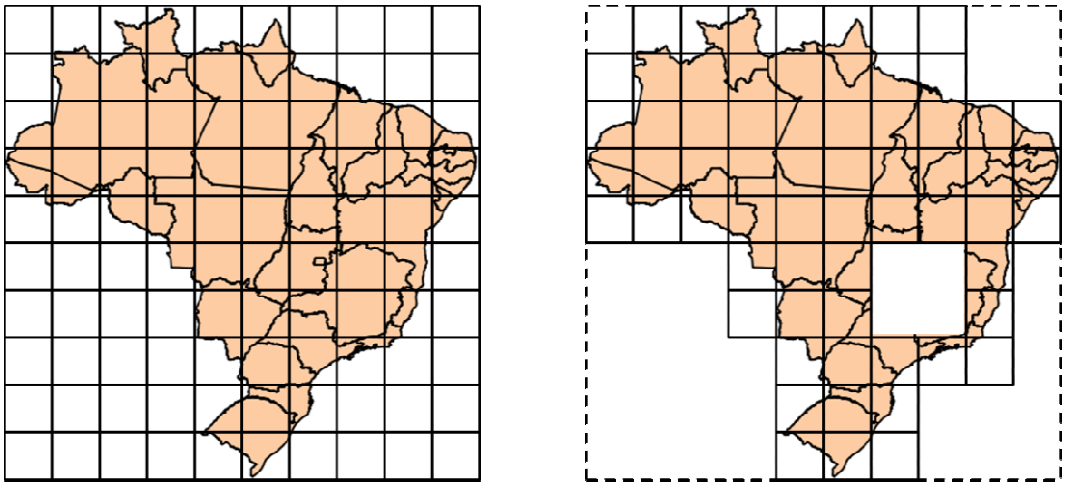
```
write (c, a, v) == Null if nature(a) == Static
synchronize (c, a) == Null if nature(a) ∈ {Static, Dynamic}
current (c,a) == read (c,a) if nature(a) ∈ {Static, Dynamic}
read (write (c, a, v), a) = v if nature(a) == Dynamic
current (write (c, a, v),a) = v if nature(a) ∈ {Dynamic, Synchronizable}
read (synchronize (write (c, a, v), a), a) == v
            if nature(a) == Synchronizable
```

Figura 3.3 - Tipo abstrato de dado - Cell.

As operações *read* e *current* realizam a leitura do atributo desejado da célula. No caso de atributos sincronizáveis, a operação *read* retorna o valor do estado passado, enquanto a operação *current* retorna o valor do atributo no estado presente. A operação *write* realiza a escrita de valores nos atributos de uma célula. No caso de atributos sincronizáveis, esta operação escreve no estado presente. A operação *synchronize*, aplicada apenas a atributos sincronizáveis, faz com que os valores dos atributos no estado presente sejam copiados para o estado passado.

3.2.2. TAD CellSpace: Espaço Celular

Um espaço celular é formado por um conjunto de células que formam a base de representação para os modelos no domínio geográfico. Espaços celulares podem ser *regulares* ou *irregulares.* Espaços regulares são formados por células cuja geometria são polígonos regulares, como retângulos ou hexágonos. Eles representam a região de interesse através de uma cobertura completa do espaço. Os espaços celulares irregulares, diferentemente, não representam a região de interesse através de uma cobertura completa do espaço, pois podem ter algumas de suas células omitidas tanto no interior quanto nas bordas da região de interesse. Cada uma dessas representações acarreta em diferentes requisitos para a realização das travessias na representação e para a computação de vizinhanças, como será mostrado mais adiante. A Figura 3.4 ilustra a diferença entre eles.



(a) Regular (b) Irregular

Figura 3.4 - Tipos de Espaços Celulares.

Um espaço celular pode ser descrito por uma matriz *n*-dimensional $M_{i,j}^{k}, k = 1..n$, onde *i* representa uma linha, *j* uma coluna e *k* uma dimensão do espaço de atributos. As células, ao longo de cada espaço de atributos, possuem o mesmo tipo de dados. As dimensões podem ser referenciadas por índices numéricos ou por nomes associados a elas.

A Figura 3.5 ilustra graficamente a representação matricial de um espaço celular. É importante ressaltar que esta matriz pode estar associada a uma área geográfica com limites bem definidos, mostrada pelo retângulo envolvente mínimo, e associada a um sistema de referência espacial.

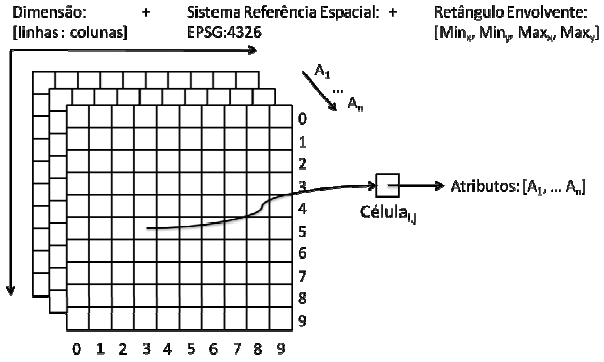


Figura 3.5 - Espaço Celular e Células.

A definição do TAD *CellularSpace* (espaço celular) é apresentada na Figura 3.6. A operação *new* constrói um novo espaço celular a partir das informações de dimensão (número de linhas e colunas) e tipo de atributos das células. As operações *retrieve*, *first*, *last* e *next* possibilitam acessar as células do espaço celular. No caso de espaços irregulares, a operação *exists* possibilita verificar a existência ou não de uma célula. A operação *synchronize* realiza a sincronização de todos os atributos sincronizáveis das células.

A operação *filter* possibilita recuperar um subconjunto do espaço celular que satisfaça a uma condição expressa por uma função do tipo *FilterFunction*. Este tipo de função, quando aplicada a uma célula, retorna verdadeiro caso ela satisfaça à condição, ou retorna falso caso-contrário. De maneira similar, a operação *sort* retorna um novo espaço celular a partir da ordenação das células do espaço original, utilizando uma função de comparação do tipo *SortFunction*. Este tipo de função deve ser capaz de estabelecer uma ordem total para o conjunto de células.

```
type  FilterFunction
syntax
   apply (Cell) → Bool

type  SortFunction
syntax
   apply (Cell, Cell) → Bool


type  Dimension

syntax

   rows (Dimension) → Int

   cols (Dimension) → Int

   size (Dimension) → Int
```

```
type  Index
syntax
   row(Index) → Int
   col(Index) → Int

type  CellSpace uses Cell, Attribute, Index, Dimension
syntax
   new ([Attribute], Dimension) → CellSpace
   dimension (CellSpace) → Dimension
   retrieve (CellSpace, Index) → Cell
   first (CellSpace) → Cell
   last (CellSpace) → Cell
   next (CellSpace, cell) → Cell
   random (CellSpace) → Cell
   regular (CellSpace) → Bool
   exists (CellSpace, Index) → Bool
   synchronize (CellSpace) → CellSpace
   filter (CellSpace, FilterFunction) → CellSpace
   sort (CellSpace, SortFunction) → CellSpace

axioms
   cs: CellSpace; c: Cell;  A: [Attribute]; d: Dimension, i: Index

   cs = new (A, d) ⇒ A ≠ ∅, size(d) > 0
   retrieve(cs, i) == Null if exists(cs, i) == false
```

Figura 3.6 - Tipo abstrato de dado - espaço celular.

3.2.3. TAD Neighborhood: Vizinhança

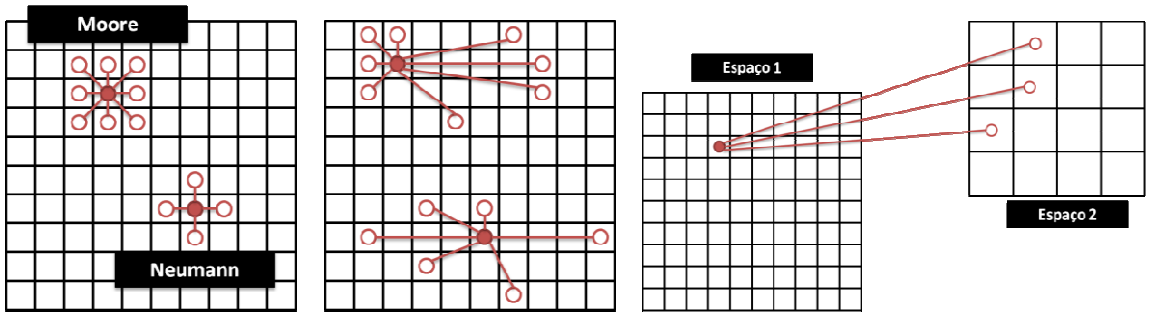
A abstração de vizinhança captura as relações que podem existir entre as células de um mesmo espaço celular ou de diferentes espaços celulares. As vizinhanças podem ser:

- *isotrópicas* ou *anisotrópicas*: vizinhanças isotrópicas respeitam a adjacência das células e se propagam da mesma forma em todas as

direções. As *anisotrópicas* são dependentes de critérios mais complexos que exigem outros conceitos como *acessibilidade*, que depende do seu contexto de uso para estabelecer uma definição funcional, por exemplo, podendo ser estabelecida com base em relações econômicas, relações sociais ou uma composição de relações;

- *estacionárias* ou *não-estacionárias*: vizinhanças também podem ser classificadas usando uma referência temporal. Aquelas que não mudam ao longo da linha de tempo são chamadas de *estacionárias*, enquanto aquelas cuja configuração muda ao longo da linha do tempo são chamadas de *não-estacionárias* (MARETTO, 2011).

Além das classificações acima, cada um dos relacionamentos em uma vizinhança pode estar associado a atributos ou pesos. A Figura 3.7 mostra exemplos desses tipos de vizinhanças.



(a) Isotrópica (b) Anisotrópica (c) Diferentes Espaços Celulares

Figura 3.7 - Tipos de vizinhanças.

A definição formal do TAD vizinhança pode ser vista na Figura 3.8. Uma vizinhança pode ser definida sobre um único espaço celular ou através de um par de espaços celulares. O operador *neighbors* fornece o acesso aos relacionamentos de uma célula, resultando em um vetor com a tripla (*célula, atributo do relacionamento, valor*). A operação *add* permite adicionar novos

relacionamentos entre as células. A operação *remove* permite remover os relacionamentos entre pares de células.

```
type  Neighborhood uses Cell, CellSpace, Attribute
syntax
   new (CellSpace, CellSpace) → Neighborhood
   neighbors (Neighborhood, Cell) → [ (Cell, Attribute, Value) ]
   add (Neighborhood, Cell, Cell, [(Attribute, Value)]) → Neighborhood
   remove(Neighborhood, Cell, Cell, [(Attribute, Value)]) → Neighborhood
axioms
   n: Neighborhood
   cs1, cs2: CellSpace
   c, c1, c2, cn: Cell
   a: Attribute
   v: Value

   n = new (cs1, cs2) ⇒ (cs1 ≠ Null)
   if (c ∈ cs1) ⇒ ∀cn ∈ neighbors(n, c) ⇒ cn ∈ cs2 iif cs2 ≠ Null
   if (c ∈ cs1) ⇒ ∀cn ∈ neighbors(n, c) ⇒ cn ∈ cs1 iif cs2 = Null
   remove(n, c1, c2, Null) ⇒ neighbors(n, c1) = ∅
```

Figura 3.8 - Tipo abstrato de dado - vizinhança.

Com os tipos definidos, podemos retornar para a caracterização dos módulos componentes da arquitetura do CellDB.

3.3. CellDB: Módulo Camada de Acesso a Dados

As fontes de dados para os modelos podem ser divididas em dois tipos: fontes que contêm os atributos das células e fontes que armazenam informações sobre vizinhanças de espaços celulares. No caso das células, as fontes de dados podem estar associadas a SGBD-R, como o PostGIS Raster (OBE et al., 2011) ou Oracle GeoRaster (ORACLE, 2012), a SGBD Matriciais, como SciDB (CUDRE-MAUROUX et al., 2009) e Rasdaman (BAUMANN et al., 1998), a Serviços Web, como o Web Coverage Service (OGC, 2012c), a arquivos

matriciais, como GeoTIFF (RITTER e RUTH, 2012), HDF (FOLK et al., 2011), NetCDF (UCAR, 2012) e, GRIB (WMO, 2012) ou uma combinação destas fontes.

Dados sobre vizinhança podem estar disponíveis a partir de formatos de arquivos para representação de grafos como GAL, GraphML (GRAPHML, 2012), GXL (HOLT et al., 2012), armazenados em SGBD específicos para grafos como Neo4J (HUNGER, 2010), ou mesmo armazenados em tabelas de um SGBD-R.

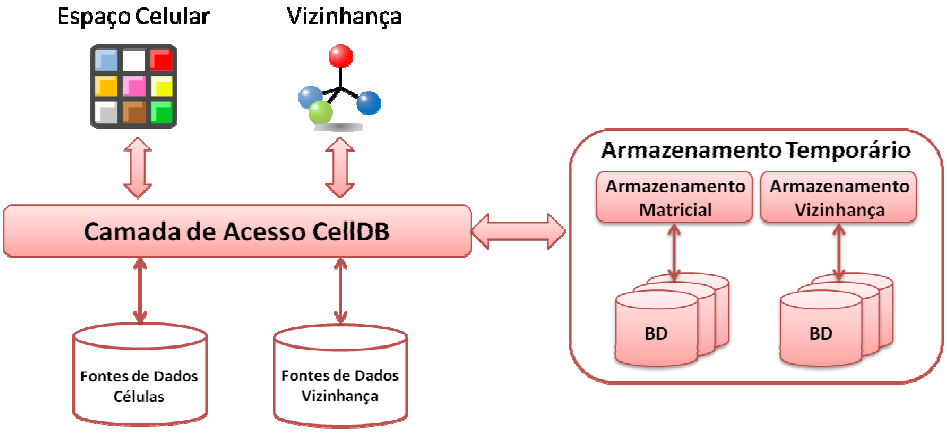


Figura 3.9 - CellDB - camada de acesso a dados especializada nos tipos espaço celular e vizinhança.

Para as ferramentas de modelagem, o CellDB funciona como uma camada de acesso especializada nos tipos de dados descritos na Seção 3.2, como destacado na Figura 3.9. É responsabilidade do CellDB recuperar os dados nos diversos tipos de fontes e prepará-los para uso, sem que a ferramenta de modelagem tenha a preocupação de como isto será feito. Da mesma forma, os resultados intermediários gerados pelas simulações poderão ser atualizados nas fontes indicadas pela ferramenta.

As simulações são sempre realizadas sobre cópias dos dados originais dos espaços celulares e vizinhanças, para preservá-los, e para permitir novas execuções a partir dos mesmos dados de entrada. Os modelos podem

referenciar todos os atributos de um espaço celular existente, ou apenas uma pequena parcela, ou podem ainda criar novos atributos dinamicamente. Para as ferramentas de modelagem deve ser transparente o fato de ser necessário um armazenamento temporário durante as simulações, seja através da criação de tabelas temporárias nos SGBDs ou de arquivos em disco.

Para as ferramentas de modelagem, a principal funcionalidade do CellDB é a capacidade de preparação do ambiente de dados usados durante as simulações. Para cada espaço celular usado em um modelo, a camada representada pela ferramenta de modelagem deverá informar:

1. Lista dos atributos contidos na definição do espaço celular que serão utilizados na simulação, juntamente com o tipo do atributo (*Static*, *Dynamic* ou *Synchronizable*);

2. Lista de atributos temporários que serão necessários para a simulação. Neste caso, além do tipo da natureza do atributo, será necessário informar o tipo de dado a ser usado em sua representação.

A Figura 3.10 ilustra a preparação do ambiente de dados até o direcionamento do local de armazenamento dos atributos do espaço celular resultantes da simulação.

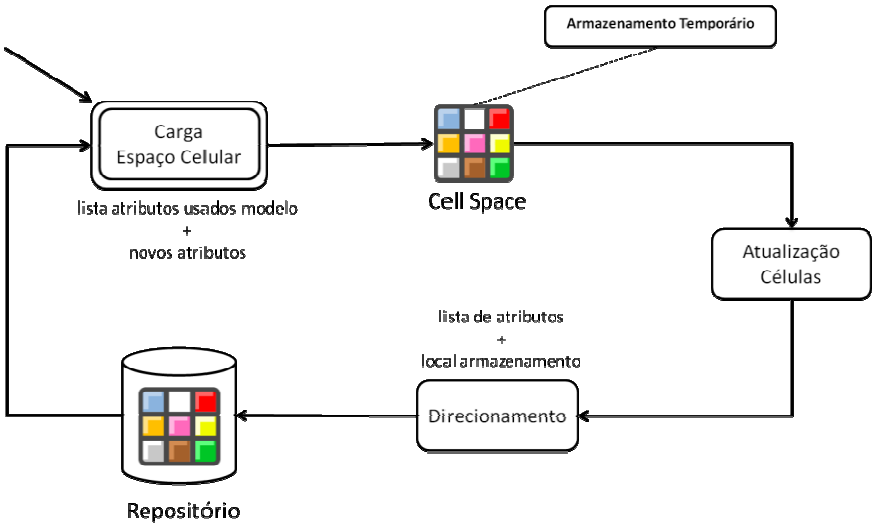


Figura 3.10 - Preparação do ambiente de dados do modelo.

3.4. CellDB: Módulo Catálogo Interno de Metadados

As definições dos espaços celulares e vizinhanças podem ser mantidas em um catálogo interno de metadados do CellDB, para facilitar o uso sucessivo dos dados em várias execuções dos modelos. Este catálogo mantém as seguintes informações:

1. Lista das fontes de dados de origem de cada espaço celular e vizinhança: para cada fonte de dados é mantido um registro com seu nome, um título a ser usado em uma eventual apresentação gráfica, uma breve descrição com maiores detalhes da fonte, o tipo de fonte de dados, o *driver* de acesso a dados a ser utilizado para recuperação e armazenamento de dados e os parâmetros de conexão com a fonte de dados;

2. Lista com as definições de espaços celulares: para cada espaço celular é mantido um registro contendo um nome que o identifique unicamente, um título a ser usado em uma eventual apresentação gráfica, uma breve descrição com maiores detalhes do espaço celular, o tipo de grade (regular ou irregular), o retângulo envolvente das células, o sistema de

referência espacial, as dimensões da grade e a lista de atributos das células. Cada atributo possui um nome, o identificador da fonte de dados de origem, o nome do conjunto de dados na fonte de dados de origem e um identificador do atributo dentro deste conjunto;

3. Lista com as definições de vizinhanças: para cada vizinhança é mantido um registro contendo um nome que a identifique unicamente, um título a ser usado em uma eventual apresentação gráfica, uma breve descrição com maiores detalhes da vizinhança, o tipo de vizinhança e a fonte de dados de origem.

O Apêndice A contém uma descrição detalhada de todas as informações mantidas no catálogo interno do CellDB, na forma de esquemas e documentos XML. Este catálogo reflete o projeto de classes mostrado na Seção 3.5.

3.5. Projeto da API do CellDB

Esta seção apresenta o projeto da API CellDB, através do uso de diagramas na notação UML (BOOCH et al., 2005).

3.5.1. Visão Geral

A API CellDB é composta por uma classe principal denominada *celldb*, que funciona como uma fachada (GAMMA et al., 1994) para as demais classes do sistema, organizadas em dois pacotes: *Catalog* e *Data Access*. O diagrama da Figura 3.11 mostra esta organização.

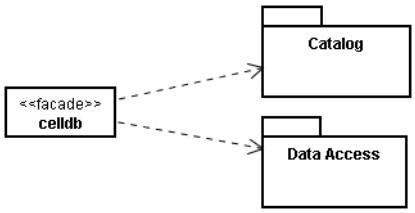


Figura 3.11 - Organização da Classes da API CellDB.

3.5.2. Classes do Catálogo Interno de Metadados

As classes do pacote *Catalog* são ligadas ao gerenciamento do catálogo interno de metadados. As seguintes classes fazem parte deste pacote: *datasource_catalog, datasource_info, cell_space_catalog, cell_space_info, cell_space_attribute_info, neighborhood_catalog* e *neighborhood_info*. A Figura 3.12 mostra o relacionamento dessas classes.

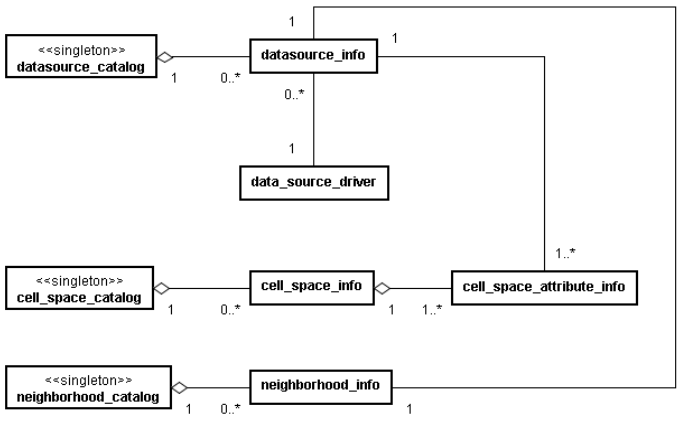


Figura 3.12 - Diagrama das classes do pacote Catalog.

A classe *datasource_catalog* contém a lista de todas as fontes de dados registradas no CellDB, que poderão ser utilizadas para a obtenção dos atributos dos espaços celulares, bem como dos dados de vizinhança. Esta classe utiliza o padrão de projeto *singleton* (GAMMA et al., 1994), possibilitando seu acesso de forma global.

As informações necessárias ao acesso de uma determinada fonte de dados são representadas pela classe *datasource_info*. As informações mantidas por objetos desta classe podem compreender desde o nome e localização de um arquivo matricial em disco, até os dados necessários para o estabelecimento de uma conexão a um SGBD, ou a um serviço Web como o WCS. Esta classe se relaciona com o *driver* de acesso a dados, que será utilizado na

recuperação ou armazenamento dos dados dos espaços celulares e vizinhanças.

A lista dos espaços celulares registrados no CellDB é mantida em um objeto da classe *cell_space_catalog*, que também utiliza o padrão de projeto *singleton*, para possibilitar seu acesso de forma global. Cada espaço celular é descrito por um objeto da classe *cell_space_info*, que contém as informações de dimensão do espaço celular, o sistema de referência espacial, o retângulo envolvente, e a lista de atributos que compõem o espaço celular. Um atributo de um determinado espaço celular é descrito por um objeto da classe *cell_space_attribute_info*, que estabelece um relacionamento com uma determinada fonte de dados registrada no CellDB.

A lista de vizinhanças armazenadas que podem ser usadas no CellDB é mantida no catálogo de vizinhanças, representado pela classe *neighborhood_catalog*. Cada vizinhança é descrita por um objeto da classe *neighborhood_info*, que se relaciona com uma determinada fonte de dados.

O diagrama de sequência da Figura 3.13 ilustra como parte das classes deste pacote é utilizada numa operação em que um novo espaço celular é registrado no CellDB. No caso da figura é suposto que o espaço celular já exista em alguma fonte de dados. Desta forma, basta que a ferramenta forneça a lista de atributos que compõem o espaço celular, juntamente com a fonte de dados dos mesmos. A classe *celldb* analisa a lista de fontes de dados contidas nesta definição, verificando se as fontes já se encontram registradas no sistema. Logo após, a definição do espaço celular é incluída no catálogo. De maneira análoga, uma vizinhança armazenada em uma fonte de dados pode ser registrada no catálogo de vizinhanças.

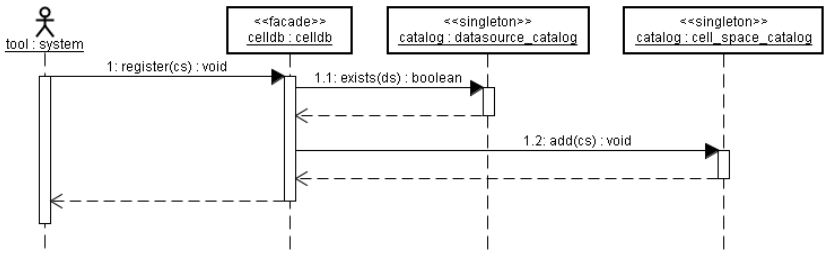


Figura 3.13 - Registrando um novo espaço celular.

3.5.3. Classes da Camada de Abstração de Dados

O pacote *Data Access* contém classes que representam os tipos abstratos de dados discutidos na Seção 3.2: *cell, cell_space* e *neighborhood*. As outras classes que compõem este pacote são: *cell_space_factory*, *cell_space_def*, *cell_space_attribute_def*, *cell_space_iterator*, *neighborhood_factory* e *neighborhood_manager*. A Figura 3.14 mostra o relacionamento dessas classes.

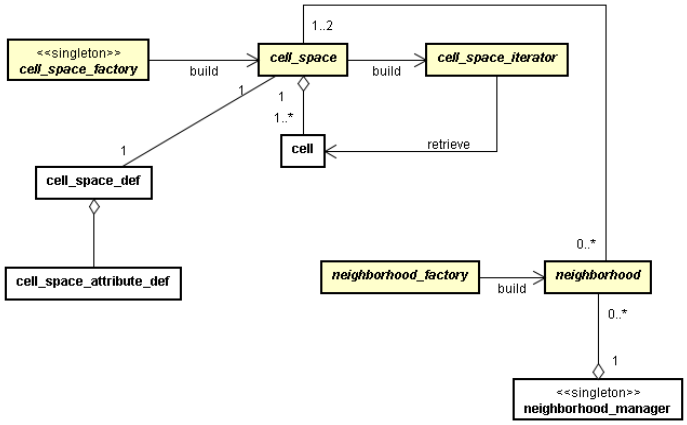


Figura 3.14 - Diagrama das classes do pacote Data Access.

Em amarelo claro são destacadas as classes abstratas *cell_space*, *cell_space_factory*, *cell_space_iterator*, *neighborhood_factory* e *neighborhood*. Elas são responsáveis por fornecer a extensibilidade necessária à API CellDB, para que implementações específicas sejam construídas, considerando os

requisitos e estratégias de armazenamento da representação interna dessas abstrações.

Espaços Celulares

A classe *cell,* no diagrama acima, representa a abstração de uma célula que pode ser manipulada diretamente pela ferramenta de modelagem. Como os modelos realizam processamentos iterativos, esta classe foi projetada como um objeto leve a ser instanciado na pilha (*stack*), funcionando como um *proxy* (GAMMA et al., 1994), para acesso aos dados reais referenciados pela célula. Para possibilitar a criação de vizinhanças armazenadas em memória secundária, cada célula possui um identificador único codificado como um número inteiro.

A abstração dos espaços celulares é representada pela classe *cell_space*. Para as ferramentas de modelagem, esta classe esconde os detalhes de organização interna dos dados do espaço celular, durante a execução de uma simulação. Além das operações básicas descritas na Seção 3.2.2, esta classe define outros métodos para possibilitar: (a) a carga de dados de espaços celulares armazenados nas diversas fontes de dados; (b) a criação de novos espaços celulares vazios; (c) a inclusão de novos espaços de atributos; (d) a remoção de um determinado espaço de atributos; (e) a cópia de atributos do espaço celular para a fonte de dados desejada.

A classe *cell_space_def* pode ser usada para a criação de definições de novos espaços celulares. Neste caso, os atributos representados por objetos da classe *cell_space_attribute_def* precisam indicar os tipos de dados usados para o armazenamento de seus valores. O CellDB suporta todos os tipos numéricos da Linguagem C++ (STROUSTRUP, 1997).

O conjunto de classes de suporte à representação dos espaços celulares pode ser usado em cenários como o mostrado no diagrama de sequência da Figura 3.15. O cenário apresentado parte do pressuposto de que a definição do

espaço celular já se encontra no catálogo do CellDB. Neste caso, a ferramenta cliente da API CellDB, representada pelo ator *tool*, precisa apenas informar o nome do espaço celular e a lista de atributos a ser utilizada na simulação. A fachada *celldb* se encarrega de localizar as informações do espaço celular, de fabricar a instância do espaço celular e invocar o método *prepare_working_copy*, para que o espaço celular esteja pronto para ser usado na simulação.

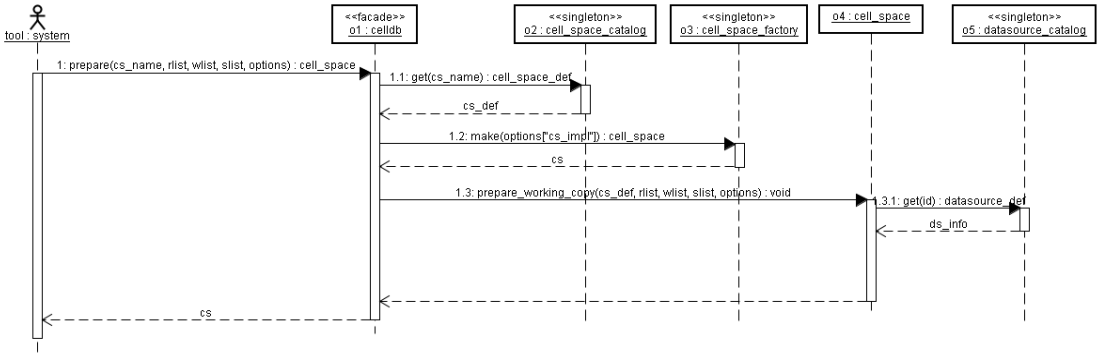


Figura 3.15 - Preparação de um espaço celular para simulação.

Iteradores

Para evitar que os clientes da classe *cell_space* tenham a necessidade de conhecer detalhes internos da organização dos espaços celulares, o CellDB utiliza o conceito de iteradores (GAMMA et al., 1994), para possibilitar o acesso iterativo às células. A Figura 3.16 ilustra três espaços celulares organizados internamente de forma distinta. O primeiro armazena as células linha a linha; o segundo armazena em blocos de células de tamanho 2x2; e o terceiro é um espaço celular irregular com blocos de tamanho 2x2. Os iteradores simplificam esta tarefa de navegação pelas células, levando em consideração a melhor forma de travessia.

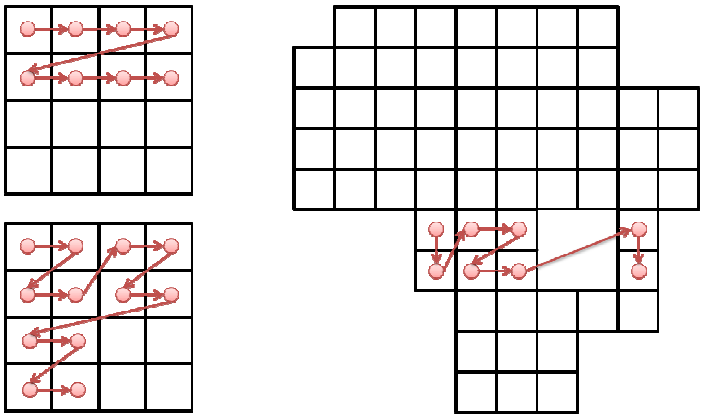


Figura 3.16 - Iteradores: desacoplando a travessia do espaço celular da organização interna das células.

A classe *cell_space* também funciona como uma fábrica de iteradores sobre as células do espaço celular subjacente. A hierarquia mostrada na Figura 3.17 corresponde aos tipos de iteradores que podem ser construídos a partir de um espaço celular.

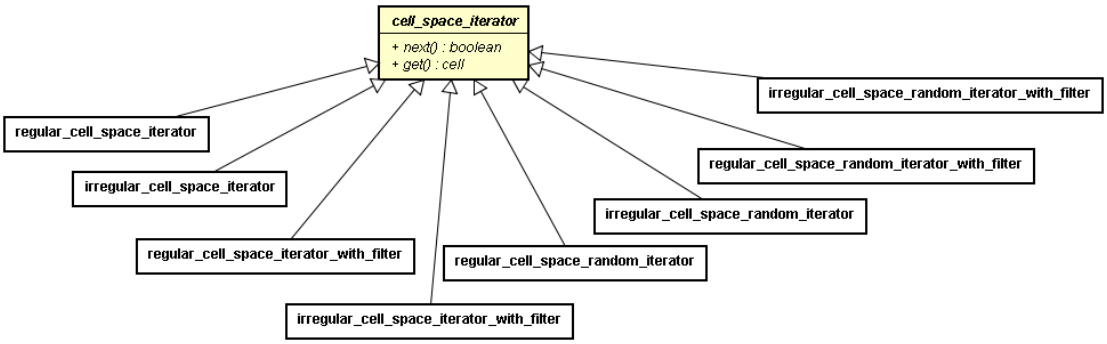


Figura 3.17 - Hierarquia de iteradores.

O seguinte tipo de função pode ser usado como parâmetro na construção dos iteradores: `bool filter_fnct_t(cell&);`

O exemplo mostrado na Figura 3.18, da função *conta_celulas_vivas*, ilustra o uso de uma função de filtro em conjunto com um iterador, para realizar a travessia das células de um espaço celular que satisfaçam a função *f1*:

```
bool f1(cell& c)
{
  return c.get_past("estado") == VIVA;
}

void conta_celulas_vivas(cell_space& cs)
{
  int contador = 0;

  cell_iterator it = cs.get_iterator(f1);

  while(it.next())
    ++contador;

 return contador;
}
```

Figura 3.18 - Obtenção e uso de um iterador.

Vizinhanças

A abstração das vizinhanças é representada pela classe *neighborhood* (Figura 3.14). Além das operações descritas na Seção 3.2.3, esta classe introduz o método *apply*, responsável por aplicar uma função informada como parâmetro à lista de células na vizinhança de um outra célula *c*. Para cada tipo de vizinhança existirá uma ou mais implementações, como mostrado no diagrama da Figura 3.19, onde se verifica a existência de pelo menos quatro tipos de vizinhanças: vizinhança oito (Moore), vizinhança quatro (Neumann), relacionamentos simples armazenados em disco (Simple) e relacionamentos com atributos associados (Weighted).

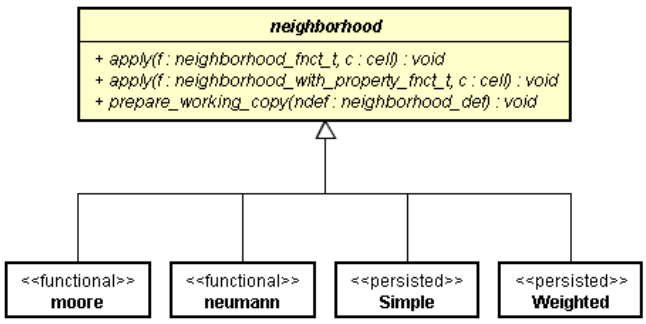


Figura 3.19 - Classe base para as vizinhanças.

Os seguintes tipos de funções podem ser usados nos métodos *apply*:

```
(1) void neighborhood_fnct_t(cell&, cell&);

(2) void neighborhood_with_property_fnct_t(cell&, cell&, double[]);
```

A função *conta_numero_vizinhos,* mostrada na Figura 3.20, ilustra o uso do primeiro tipo de função, para acessar a vizinhança de uma determinada célula:

```
int num_vizinhos = 0;

void f1(cell& c, cell& viz)
{
  ++num_vizinhos;
}

int conta_numero_vizinhos(cell& c, neighborhood& n)
{
  num_vizinhos = 0;

  n.apply(f1, c);

 return num_vizinhos;
}
```

Figura 3.20 - Acesso às células em uma vizinhança

A classe *neighborhood_manager* (Figura 3.14) é um *singleton* usado para facilitar o acesso e gerenciamento das vizinhanças associadas aos espaços celulares.

3.6. Um Artefato Computacional Demonstrativo da Arquitetura Proposta: Projeto e Implementação da API CellDB

Para demonstrar as propostas apresentadas nas seções anteriores, foi projetada e implementada uma versão do *middleware* CellDB, cujo projeto e algumas questões relativas a sua implementação são mostrados a seguir. Este artefato computacional foi gerado com o objetivo de promover uma demonstração de base empírica dos conceitos apresentados neste trabalho, possibilitando a construção dos experimentos apresentados no Capítulo 4.

3.6.1. Projeto da Arquitetura

Como a API CellDB é baseada em dois *containers* de manipulação de dados, *cell_space* e *neighborhood*, dividimos nossa arquitetura em duas partes. A primeira, mostrada na Figura 3.21, lida com a manipulação das células, onde observa-se a existência:

- de um mecanismo de *cache* (*cell_cache*), usado para possibilitar a manipulação de dados em volume maior que a capacidade da memória primária;
- um componente denominado *I/O Task Manager* responsável por gerenciar as operações de entrada e saída de dados;
- uma organização para o armazenamento temporário das células, considerando a necessidade de representar o estado passado e presente dos atributos;
- metadados associados à instância do espaço celular, como informações da organização dos arquivos temporários, forma e tamanho dos blocos do *cache*, tipos de dados de cada espaço de atributo do espaço celular e o nome usado para referenciar cada um dos atributos.

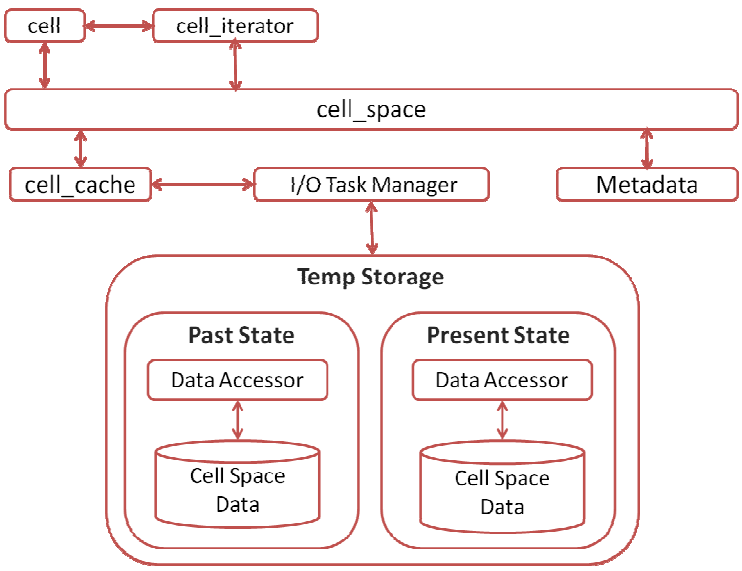


Figura 3.21 - Arquitetura CellDB - acesso e manipulação das células.

A segunda parte da arquitetura lida com a vizinhança, podendo ser dividia em vizinhanças computadas dinamicamente, chamadas de funcionais (*functional*), e vizinhanças armazenadas (*persisted*) (Figura 3.22). Em particular, no caso das vizinhanças armazenadas, decidimos empregar sistemas NoSQL da classe *Chave-Valor* para o gerenciamento destes dados. A decisão de projeto leva em conta os seguinte fatos:

- evitar gargalos na comunicação entre cliente e servidor;
- ter maior facilidade para representação dos relacionamentos das células;
- por este tipo de ferramenta fornecer implementações em disco de *Árvores-B*$^{+}$ e tabelas *hash* que possibilitam realizar pesquisas de acesso aleatório, necessárias durante o processamento das vizinhanças das células;
- facilidade de realizar travessias sequenciais;
- maior controle sobre o mecanismo de *cache* interno destes sistemas.

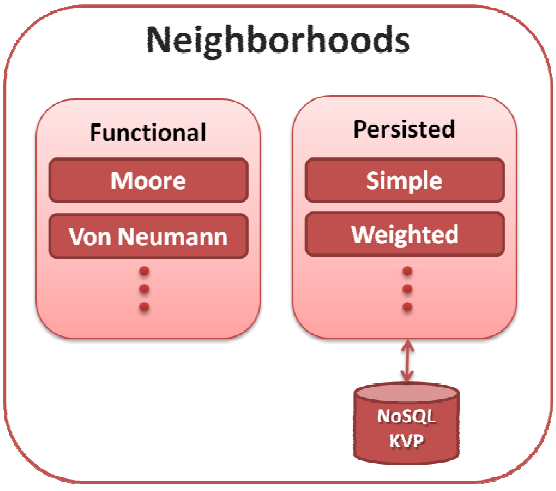


Figura 3.22 - Vizinhança.

A seguir, apresentamos maiores detalhes sobre cada parte da arquitetura das Figuras 3.21 e 3.22.

3.6.2. Estratégia de Armazenamento Temporário - I: Arquivos Matriciais

Esta estratégia de implementação do armazenamento temporário da Figura 3.21 parte do requisito de não ser necessária a instalação de softwares adicionais, como SGBDs. A relativa complexidade de configuração desses sistemas foge ao perfil dos usuários convencionais da maioria das ferramentas de modelagem. Sendo assim, é necessária uma versão do CellDB que exija o menor esforço possível por parte dos usuários finais das ferramentas em relação ao gerenciamento de dados.

As células são mantidas em arquivos matriciais criados temporariamente durante a execução das simulações. Cada dimensão do espaço de atributos de um espaço celular é mapeada para um arquivo separado (Figura 3.23). Essa separação possibilita utilizar tipos de dados mais específicos, para cada espaço de atributo. Dessa forma, evita-se aumentar desnecessariamente o volume de dados do armazenamento temporário. Além disso, essa forma de trabalho possibilita que os arquivos relativos a atributos de leitura ou de escrita possam ser mantidos em discos separados, como forma de explorar o paralelismo do hardware quando possível.

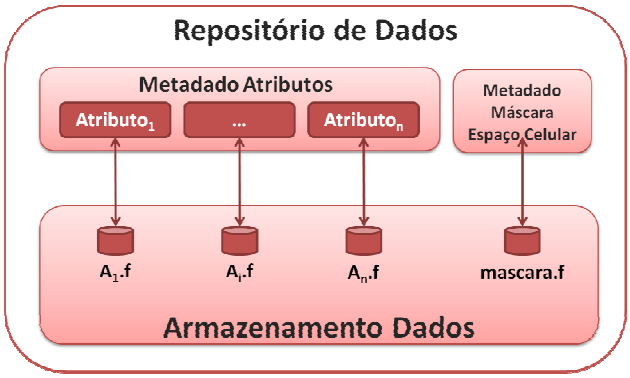


Figura 3.23 - Visão Interna do armazenamento das células em formatos de arquivos matriciais.

Várias condições especiais precisam ser levadas em consideração para realizar de forma eficiente a preparação dos dados de um espaço celular:

- Atributos estáticos, que são somente de leitura, podem ser mantidos entre as execuções das simulações, evitando cópias desnecessárias a cada execução. No caso de atributos que já estejam num formato de arquivo matricial, a cópia pode ser até mesmo evitada;

- Atributos dinâmicos, que são de leitura e escrita, são mapeados para arquivos criados dinamicamente toda vez que o ambiente de dados é preparado. Estes arquivos são removidos no final do tempo de vida do espaço celular;

- Atributos sincronizáveis necessitam manter dois arquivos temporários, uma vez que exigem o dobro do espaço de armazenamento dos demais tipos de atributos. Da mesma forma que os atributos dinâmicos, eles são criados durante a preparação do ambiente de dados e removidos no final;

- Novos atributos, que precisam ser usados no modelo, podem ser criados dinamicamente; neste caso, cada um deles é mapeado para um arquivo em disco, seguindo as regras mencionadas anteriormente.

3.6.3. Estratégia de Armazenamento Temporário - II: SGBDs Matriciais

No caso de usuários que trabalham com ambientes compartilhados através de SGBDs, o armazenamento temporário pode ser realizado diretamente sobre estes sistemas. No caso particular da extensão PostGIS Raster, o armazenamento é realizado como mostrado na Figura 3.24. Cada atributo do espaço celular é mapeado para uma tabela temporária, para facilitar operações iterativas de atualização. Essa organização é a que melhor se adequa à forma

como esta extensão foi projetada. A discussão sobre os tipos de atributos feita na Seção 3.6.2 também se aplica a esta forma de armazenamento.

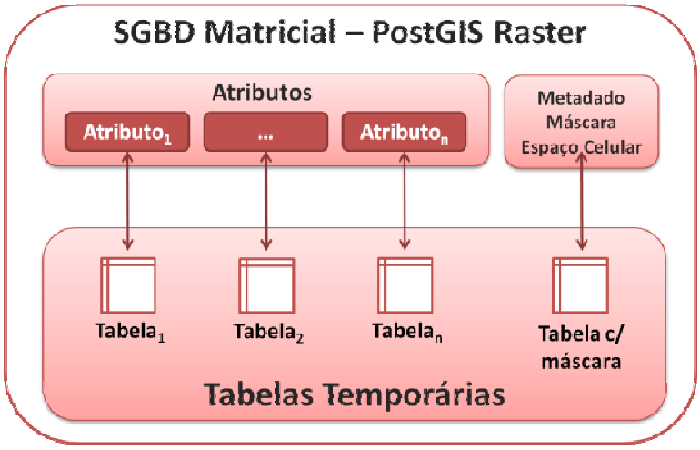


Figura 3.24 - Visão interna do armazenamento das células em SGBDs com suporte matricial.

3.6.4. Representação de Espaços Celulares Irregulares

Para espaços celulares irregulares contendo vários espaços de atributos, o CellDB utiliza uma máscara de bits para auxiliar na definição das células pertencentes ao espaço (Figura 3.25). Essa máscara evita que os iteradores tenham a necessidade de avaliar cada um dos atributos de uma célula durante a travessia do espaço celular, ajudando a aumentar a eficiência desses iteradores.

A máscara de bits pode ser gerada pelo próprio CellDB durante a criação de um espaço celular no qual um dado matricial ou vetorial tenha sido utilizado na definição dos limites das células; ou pode ser informada durante o registro do espaço celular no catálogo interno de metadados.

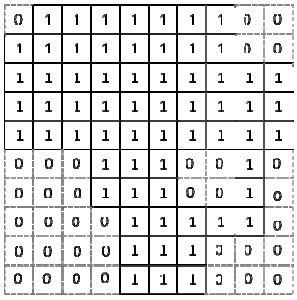


Figura 3.25 - Máscara de bits de um espaço celular irregular.

O espaço de armazenamento adicional requerido por esta máscara é desprezível quando comparado ao volume de dados ocupados pelas células e relações de vizinhança. Por exemplo, em um espaço celular de 4.096 linhas por 4.096 colunas, com dois atributos em ponto flutuante (8 bytes cada) e uma vizinhança contendo em média 4 elementos, o volume de dados ocupado pelo espaço celular é de aproximadamente:

$$numero\ de\ bytes = 4096 \times 4096 \times (2 \times 8 + 4 \times 4) = 536.870.912$$

neste caso, o volume ocupado pela máscara é de aproximadamente 2.097.152 bytes, ou seja, 0,39% do volume de dados do espaço celular.

Para o caso de espaços celulares formados por um único espaço de atributos, os valores *dummy* (ou *missing data)*, quando informados, são considerados na definição das células e na travessia dos iteradores.

3.6.5. Cache e Gerenciamento de E/S

O *cache*, mostrado na Figura 3.21, é organizado em blocos de dados separados por atributos. Esta organização:

- evita alocações por objeto (célula), algo computacionalmente caro neste contexto;
- facilita a criação de um índice que ocupe pouco espaço de armazenamento em memória para as células e seus atributos;

- possibilita manter apenas os atributos que estejam sendo usados localmente em um processamento.

O papel do *cache* nesta arquitetura é de possibilitar que alguns padrões de acesso sejam realizados de forma eficiente e que o dado não precise necessariamente ser todo carregado na memória primária. Para possibilitar que no futuro novas políticas de substituição de dados no *cache* sejam incorporadas ao CellDB, esta abstração foi projetada de forma extensível. Nesta implementação em particular, utilizamos políticas clássicas como *FIFO* e *LRU* (CHOU; DEWITT, 1985), que veremos em execução nos experimentos do Capítulo 4.

Com o surgimento de máquinas *multicore*, o emprego de paralelismo nos programas têm sido cada vez mais importante. O componente denominado *I/O Task Manager*, na Figura 3.21, tem grande interação com o *cache* de dados, sendo responsável por gerenciar as operações de entrada e saída de dados em uma *thread* de execução própria.

3.6.6. Vizinhanças em sistemas NoSQL

Vizinhanças que demandam armazenamento temporário são mantidas em sistemas NoSQL da classe Chave-Valor como Oracle Berkeley DB (OLSON et al., 1999) e LevelDB (DEAN; GHEMAWAT, 2012). Cada tipo de vizinhança possui uma organização particular de armazenamento:

- Vizinhanças simples são armazenadas como pares de chave-valor. As chaves correspondem aos identificadores numéricos dados a cada célula do espaço celular. O valor corresponde à lista de adjacências da célula, formada por uma lista numérica de identificadores (Figura 3.26);

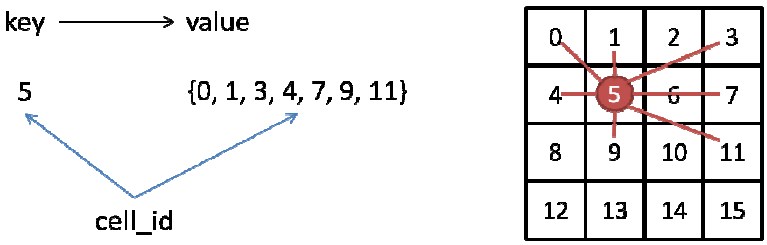


Figura 3.26 - Armazenamento de vizinhanças simples.

- Vizinhanças entre células de um mesmo espaço celular, com atributos associados, são armazenadas como mostrado na Figura 3.27. Para cada relacionamento é mantido o identificador da célula relacionada, o número de atributos no relacionamento e a lista de atributos do relacionamento;

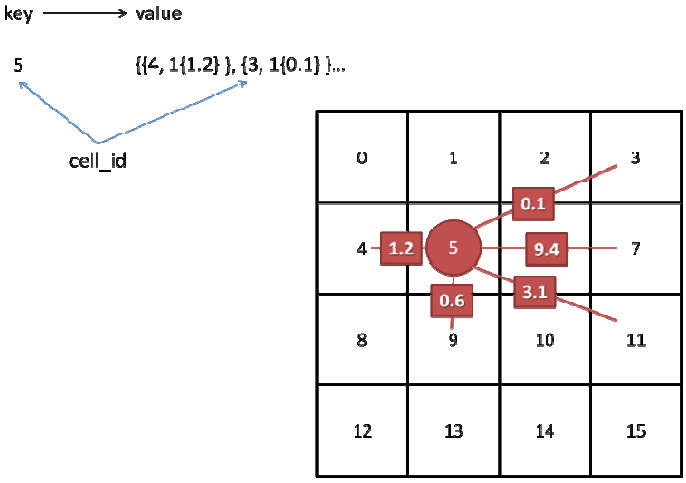


Figura 3.27 - Armazenamento de vizinhanças com atributos associados.

- Vizinhanças entre células de espaços celulares dão origem a dois arquivos de dados, um para cada lado do relacionamento. Esta forma de representação possibilita que os atributos das células sejam diferentes em cada direção do relacionamento.

3.6.7. Algumas Considerações Sobre a Implementação

A arquitetura discutida nas subseções anteriores foi implementada sobre a nova geração da biblioteca TerraLib, a TerraLib 5.0 (INPE, 2012). A Figura 3.28 apresenta a parte da arquitetura relacionada à manipulação das células (Figura 3.21), implementada utilizando a TerraLib como apoio às tarefas de manipulação dos dados.

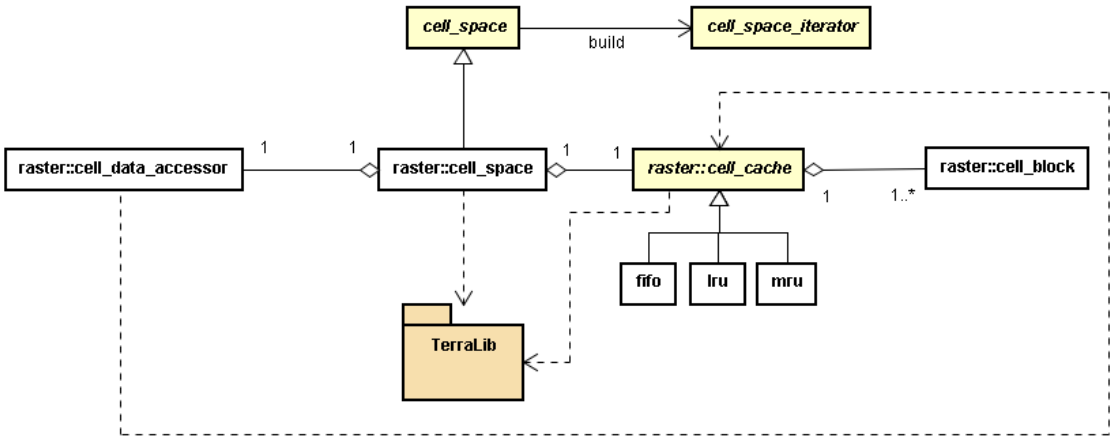
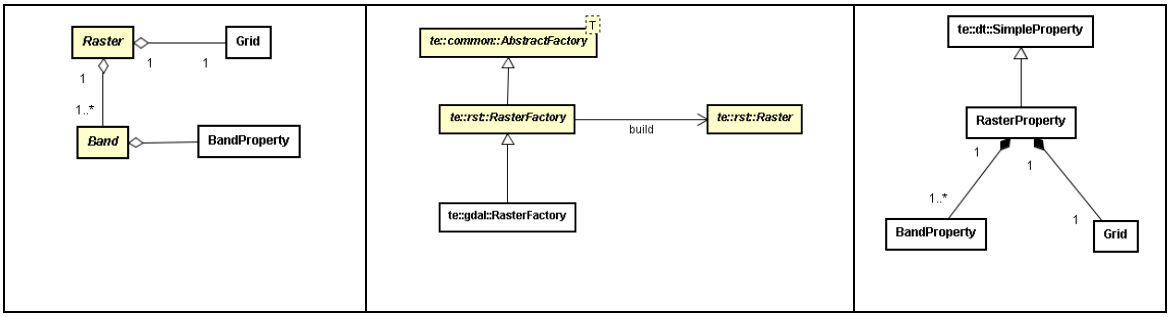


Figura 3.28 - Implementação da API CellDB sobre a TerraLib.

A escolha da TerraLib facilitou a construção do protótipo demonstrativo para o CellDB, uma vez que a biblioteca fornece as abstrações necessárias para manipulação de dados geoespaciais através de dois de seus módulos: *Raster* e *Data Access* (INPE, 2012). O primeiro módulo, *Raster*, fornece a interface básica das classes de manipulação de dados geográficos na forma de dados matriciais especializados, como imagens de sensoriamento remoto. As implementações deste módulo, denominadas *drivers*, são especializadas na manipulação, por exemplo, de cada formato de imagem e sistema de armazenamento. A Figura 3.29 mostra um diagrama com as principais classes deste módulo.

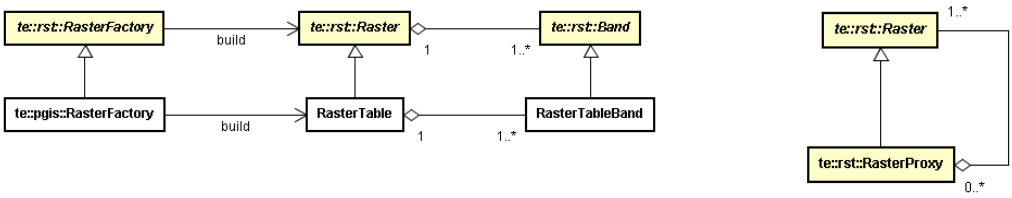


(a) (b) (c)

Figura 3.29 - Classes Módulo Raster TerraLib: (a) Composição de um raster; (b) Fábricas de raster; (c) Definição de propriedades de um raster.

A classe *cell_space* e o *cache* de células, mostrados na Figura 3.28, foram implementados sobre esta *API Raster*. Na manipulação dos arquivos temporários foi utilizado o *driver raster* da TerraLib, construído com a biblioteca GDAL (WARMERDAM, 2010). Para a prototipação da estratégia de armazenamento temporária sobre tabelas matriciais do PostGIS Raster, foi desenvolvido um *driver* específico, mostrado na Figura 3.30a.

Para simplificar a manipulação dos espaços de atributos das células, assim como o desenvolvimento do *cache*, sobre vários arquivos ou tabelas, foi desenvolvida uma classe denominada *ProxyRaster*. Esta classe fornece uma visão única sobre um conjunto de *rasters* (Figura 3.30b).



(a) (b)

Figura 3.30 - (a) Driver PostGIS Raster; (b) ProxyRaster

O segundo módulo da TerraLib, denominado *data access*, possibilita que as aplicações manipulem dados espaciais em diferentes tipos de fontes de dados,

abstraindo desde os tradicionais SGBDs até os Serviços Web da OGC. Este módulo é composto pelas seguintes abstrações (Figura 3.31): *data source*, *dataset*, *data source catalog*, *dataset type*, *query* e *data source catalog loader* (INPE, 2012). Grande parte das classes desta camada são abstratas, isto é, elas definem apenas a interface que deve ser implementada por *drivers* específicos que entendam dos detalhes de cada tipo de fonte.

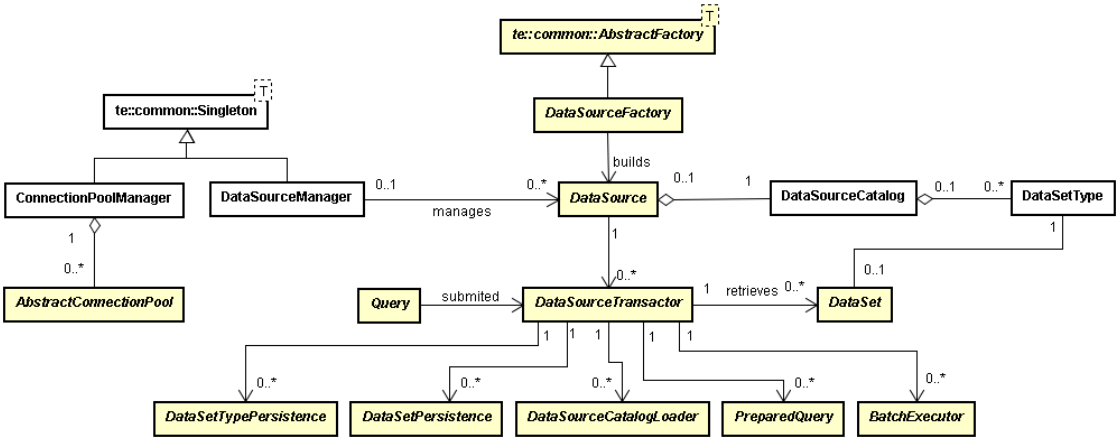


Figura 3.31 - Diagrama UML das classes de acesso a dados da TerraLib 5.

No protótipo do CellDB, esta camada da TerraLib foi utilizada nos métodos de preparação do ambiente de dados das simulações, assim como para copiar os resultados intermediários gerados por elas para o repositório de dados final. Esta estratégia possibilitou uma rápida prototipação da arquitetura proposta para o CellDB.

3.7. Considerações Finais

A versão do protótipo construída para este trabalho possui algumas limitações. O componente de gerenciamento de E/S de forma paralela (*multithread*) não foi implementado. Uma implementação sequencial acoplada ao *cache* foi a opção, o que simplificou o desenvolvimento e não comprometeu a avaliação das principais ideias apresentadas.

Em relação às classes de vizinhança, implementamos apenas a vizinhança funcional de Moore (vizinhança 8) e o suporte a vizinhanças simples armazenadas no Oracle Berkeley DB. Não foram incluídos métodos para alteração dinâmica das vizinhanças armazenadas nem *drivers* de acesso aos formatos de grafo mencionados na Seção 3.3.

No próximo Capítulo apresentamos experimentos realizados com o protótipo do CellDB, para avaliação de seu desempenho diante de grandes espaços celulares.

4 PROVA DE CONCEITO: AVALIANDO O DESEMPENHO DE TRÊS MODELOS ESPACIALMENTE EXPLÍCITOS COM O USO DO CellDB

Neste capítulo, apresentamos uma análise parcial de desempenho para uma implementação do CellDB, observando duas características principais associadas a sua proposta: (1) a possibilidade de tratar com grandes espaços celulares, portanto em problemas em que o volume de dados é relevante e, (2) uma avaliação da estratégia de *cache* proposta para o CellDB e o impacto das políticas adotadas. Esta avaliação é feita tomando como base três modelos que apresentam dinâmicas espacialmente explícitas: *o Jogo da Vida* (GARDNER, 1970), *o Modelo de Segregação de Schelling* (SCHELLING, 1971) e um *Modelo de Uso e Cobertura da Terra* (LUCC) com base em modelos tradicionais de alocação espacial de demanda (DE KONING et al., 1999). Neste trabalho, um modelo simplificado tendo por base o trabalho de AGUIAR (2006) foi utilizado. Estes modelos apresentam dinâmicas espacialmente explícitas e possuem características de acesso a dados que constituem uma amostra representativa dos padrões de acesso encontrados em diversos modelos da literatura. A descrição destes modelos, da metodologia de testes e os resultados dos experimentos são apresentados e analisados neste capítulo.

4.1. Metodologia para Construção dos Testes

Para esclarecer como foram projetadas as baterias de testes, apresentamos primeiro uma discussão sobre cada modelo e sua implementação.

4.1.1. Jogo da Vida

O primeiro modelo discutido é o jogo da vida, ou *game of life*, originalmente proposto por John Conway (GARDNER, 1970). Este modelo tem como objetivo simular o processo de nascimento e morte de organismos em uma sociedade. Cada célula de um espaço celular possui um estado vivo ou morto, que indica a presença ou ausência de um indivíduo. A simulação se inicia com uma determinada configuração espacial de células vivas e computa a evolução

desta população com o passar do tempo. Cada célula atualiza o seu estado usando regras determinísticas baseadas nos estados dos seus oito vizinhos:

- uma célula viva morre de solidão se possuir menos de dois vizinhos vivos;
- uma célula viva morre por superpopulação se possuir mais de três vizinhos vivos;
- uma célula morta se transforma em viva se possuir exatamente três vizinhos vivos;
- uma célula viva mantém o seu estado se possuir dois ou três vizinhos vivos.

A Figura 4.1 ilustra este processo, que ocorre em paralelo no tempo para cada célula. Assim, a alteração de estado de uma determinada célula não pode afetar o estado das células vizinhas em um mesmo instante de tempo.

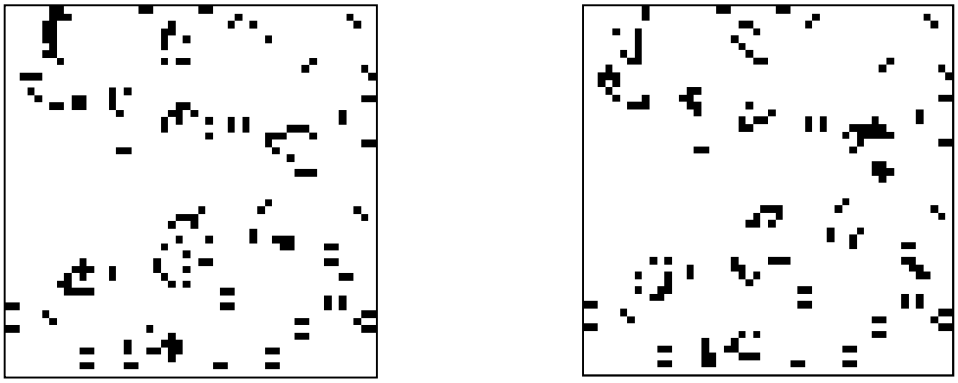


Figura 4.1 - Jogo da Vida.

Baseado no grande sucesso deste modelo, diferentes autores usaram as suas ideias básicas para desenvolver modelos computacionais nas mais diversas áreas da ciência. Exemplos incluem tanto processos ambientais, como propagação de fogo e escoamento de água superficial (LI; MAGILL, 2001; PARSONS; FONSTAD, 2007), quanto processos humanos, como tráfego e

crescimento urbano (ROSENBLUETH; GERSHENSON, 2011; BATTY et al., 1999).

Para o propósito de avaliação do CellDB, é a manipulação dos dados no espaço celular que nos interessa. O modelo do jogo da vida ilustra um cenário típico onde (i) as células necessitam da manutenção de dois estados (*presente* e *passado*); (ii) da leitura apenas de variáveis do estado passado e escrita apenas no presente; (iii) do uso de vizinhanças isotrópicas e (iv) da necessidade de realização de uma travessia por todas as células de forma iterativa.

O pseudocódigo do jogo da vida mostrado no Quadro 4.1 ilustra o algoritmo usado na sua implementação. Os trechos de código nos Quadros B.1 e B.2 do Apêndice B apresentam a implementação completa deste modelo diretamente sobre a API do CellDB.

```
estado ← Atributo("estado")

dimensão ← 1024x1024

cs ← Espaço_Celular(dimensão, estado)

para cada Célula c ∈ cs faça

   e ← sorteio-aleatório(VIVO, MORTO)

   c.escreva(estado, e);

fim para

cria_vizinhança_moore(cs) // vizinhança 8

num_iteracoes ← 10

para i ← 1 até num_iteracoes faça
   para cada Célula c ∈ cs faça
      vz ← vizinhança(c)

      n ← número_células_vivas(vz)

      es ← c.leia(estado)

       se((es == VIVO) e ((n > 3) ou (n < 2))) então
          c.escreva(estado, MORTO)
```

```
        senão((es == MORTO) e (n == 3))
           c.escreva(estado, VIVO)
        fim se

    fim para

    sincronize(cs)

fim para
```

Quadro 4.1 - Pseudocódigo do Jogo da Vida

4.1.2. O Modelo de Segregação de Schelling

O segundo modelo usado nos testes é o de Thomas Schelling, usado, originalmente, para estudo de processos de segregação racial no espaço (SCHELLING, 1971). Neste modelo, células brancas, vermelhas e pretas são distribuídas sobre um espaço celular. As vermelhas e pretas representam indivíduos nas suas moradias, enquanto que as brancas são áreas desocupadas. Qualquer indivíduo tolera viver em uma vizinhança que possua indivíduos da cor oposta, mas apenas até o limite de metade da sua vizinhança. Caso algum indivíduo esteja em minoria na sua vizinhança, ele decide se mudar para uma nova célula vazia na qual ele esteja em maioria ou igualdade na nova vizinhança. A cada instante de tempo, um indivíduo verifica a sua satisfação e decide mudar ou não. Este processo se repete iterativamente até que todos os membros da população estejam satisfeitos. A Figura 4.2 ilustra o estado inicial (esquerda) e a situação final na qual o modelo se encontra estável (direita).

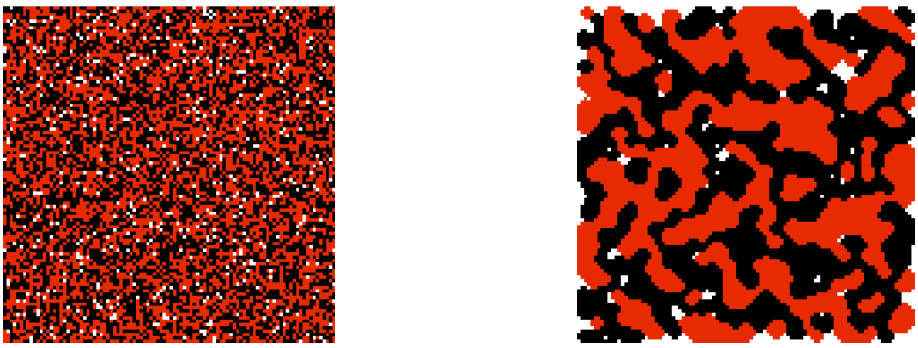


Figura 4.2 - Modelo de segregação de Schelling.

A dinâmica de Schelling, demonstrada com seu modelo, tem sido citada como uma forma de explicar as variações que são encontradas em elementos considerados como portadores de significativas diferenças como - de gênero, idade, raça, etnia, língua, orientação sexual e religião. Seguindo as ideias centrais de Schelling, com algumas adaptações, todo um conjunto de modelos para tratar dinâmicas de segregação residencial e economia urbana foram desenvolvidos (BATTY, 2007; FEITOSA et al., 2011, 2012).

Para o propósito de avaliação do CellDB, são as características do acesso aos dados no espaço celular que nos interessam. Este modelo exemplifica um padrão de acesso no qual (i) as células necessitam apenas da manutenção de um estado (presente); (ii) do uso de vizinhanças isotrópicas e (iii) da necessidade de realização de uma travessia de forma aleatória pelas células. O pseudocódigo mostrado no Quadro 4.2 ilustra o algoritmo usado na implementação deste modelo. Os trechos de código nos Quadros B.3 e B.4 do Apêndice B apresentam a implementação deste modelo diretamente sobre a API do CellDB.

```
taxa_desocupação ← 0.03

limiar_satisfação ← 4

cor ← Atributo("cor")

cs ← Espaço_Celular(1024 x 1204, cor)

para cada Célula c ∈ cs faça

   valor ← sorteio-aleatório(0.0, 1.0)

   se(valor < taxa_desocupação) então
      c.escreva(cor, BRANCO)
   senão se(valor < ((1.0 + taxa_desocupação) / 2.0))
      c.escreva(cor, VERMELHO)
   senão
      c.escreva(cor, PRETO)
   fim se

fim para

função Infeliz(c:Cor,  cel:Célula)
```

```
   se(cel.leia(cor) != c)
      retorna falso
   fim se

   vz ← vizinhança(cel)

   n ← conta_vizinhos_cor(vz, c)

   retorna (n < limiar_satisfação)

fim função


função Vaga(c:Cor,  cel:Célula)

   se(cel.get_present(cor) <> BRANCO)
      retorna falso
   fim se

   vz ← vizinhança(cel)

   n ← conta_vizinhos_cor(vz, c)

   retorna (contador >= limiar_satisfação)

fim função


função Troca(cs:Espaço-Celular, c:Cor)

   célula_antiga = sorteia_célula(cs, c, Infeliz)

   se(!celula_antiga == NULO)
      retorna falso
   fim se

   célula_nova = sorteia_célula(cs, c, Vaga)

   se(!celula_nova == NULO)
      retorna falso
   fim se

   troca_valores(celula_nova, celula_antiga)

   retorna verdadeiro

fim função


cria_vizinhança_moore(cs)  // cria vizinhança 8

num_iteracoes ← 100

para i ← 1 até num_iteracoes faça
```

```
   r1 ← Troca(cs, VERMELHO)

   r2 ← Troca(cs, PRETO)

   se(não(r1 ou r2))
      interrompe
   fim se

fim para
```

Quadro 4.2 - Pseudocódigo do modelo de segregação

4.1.3. Modelo LUCC – Dinâmica de Mudança de Uso e Cobertura da Terra

O terceiro modelo usado nos testes é uma versão simplificada de uma visão de dinâmica de Mudança de Uso e Cobertura da Terra (*Land Use and Cover Change - LUCC*) apoiada na ideia de cálculo de demanda potencial e sua alocação espacial. A implementação escolhida representa uma simplificação do modelo completo proposto por Aguiar (2006). Diferentes propriedades estabelecidas espacialmente são usadas para computar o potencial de uma célula ser desmatada, como por exemplo distância para a estrada mais próxima, porcentagem de área protegida e média de desmatamento das células vizinhas. O modelo então aloca uma quantidade pré-determinada de desmatamento de acordo com os potenciais de cada célula. Vários modelos da literatura seguem esta estrutura de definição para estudar processos de mudança de uso da terra (BERRY et al. 96; PONTIUS et al., 2001; SOARES-FILHO et al., 2002; VELDKAMP; FRESCO, 1996; VERBURG et al. 2002). O pseudocódigo mostrado no Quadro 3 ilustra o algoritmo usado na implementação deste modelo. O trecho de código mostrado no Quadro 5 do Apêndice B apresenta a implementação deste modelo diretamente sobre a API do CellDB.

```
demanda_anual ← 30000  // desflorestamento anual

cs ← Espaço_Celular("área-de-estudo")

cria_vizinhança_moore(cs)  // cria uma vizinhança 8
```

```
num_iteracoes ← 2

para i ← 1 até num_iteracoes faça

   total_potencial ← 0

   para cada Célula c ∈ cs faça

      total_potencial ← total_potencial + calcula_potencial(c)

   fim para

   alocacao ← demanda_anual / total_potencial

   para cada Célula c ∈ cs faça

      aloca_desflorestamento(c, alocacao)

   fim para

fim para
```

Quadro 4.3 - Pseudocódigo do modelo simplificado LUCC

4.1.4. Infraestrutura de hardware e software utilizada para os Testes

Todos os modelos foram implementados na linguagem C++, em uma arquitetura alvo de 64-bits. Para execução das baterias de testes, foi utilizada uma máquina com o sistema operacional Windows-7 64-bit, 8 GB de memória RAM, 500 GB de disco com velocidade de rotação de 7.200 rpm e processador Intel Core i7. Excluindo-se a memória reservada para uso do sistema operacional e aplicativos nativos, a quantidade de memória RAM livre disponível na máquina é de aproximadamente 6,8 GB. Esta máquina foi escolhida por representar um equipamento típico disponível para grande parte dos pesquisadores e alunos que têm utilizado modelos computacionais em seus estudos no INPE.

4.1.5. Definição e Descrição das Baterias de Testes

Dada a variedade de tipos básicos de dados suportados no CellDB, para evitar um número exponencial de casos de teste nas baterias projetadas, fixamos o tipo básico de dado usado para representação dos atributos das células em

todos os experimentos em um número em ponto flutuante de 64-bit. Este tipo numérico é o mais utilizado nos modelos reais para os quais a arquitetura do CellDB foi projetada. Também foi a forma encontrada para facilitar a geração de dados sintéticos com volumes de dados consideráveis. A seguir uma descrição das baterias de testes é apresentada e os resultados obtidos são apresentados e comentados.

Tabela 4.1 - Preâmbulo dos modelos usados nos experimentos.

Modelos	Justificativa
Joga da Vida	exemplos de autômato celular / atributos sincronizáveis
Segregação	estudos de mobilidade / acesso aleatório células
LUCC	modelo padrão de uso da terra / manipulação de múltiplos atributos

Bateria 1: Desempenho do CellDB relacionado a Grandes Volumes de Dados

Esta bateria de testes verifica o desempenho do CellDB para a manipulação de grandes volumes de dados em computadores pessoais. Ela utiliza espaços celulares cujos volumes de dados são bem maiores do que a quantidade de memória livre do ambiente de hardware e software usado para os experimentos. Para isso, utilizamos o Jogo da Vida e o modelo simplificado LUCC. Durante a execução do Jogo da Vida é necessário manter os estados presente e passado dos atributos das células. Com isso, o volume de dados manipulado no teste é o dobro do tamanho ocupado por uma única dimensão do espaço de atributos. No caso de um espaço celular com dimensões de 32.768x32.768, o volume de dados manipulado é de 16 GB.

A implementação do modelo LUCC manipula cinco atributos para cada célula. São eles: distância mínima a centros urbanos, distância mínima a mercados, porcentagem de área protegida, porcentagem de desflorestamento e potencial

de mudança. Destes cinco atributos, os três primeiros são somente para leitura e os outros dois para leitura e escrita. Portanto, o volume de dados manipulado é cinco vezes o valor de uma única dimensão do espaço de atributos. Em um espaço celular de 32.768x32.768, com cada célula contendo cinco atributos em ponto flutuante de 64-bit, o volume total é 40GB.

Para demonstrar a viabilidade do uso do CellDB, em cada simulação dos modelos, realizamos cinco execuções com três iterações cada, tomando o tempo médio gasto com cada iteração. Os valores para cada tamanho de espaço celular e modelo são mostrados na Tabela 4.2. Em vermelho destacamos os espaços celulares cujo volume de dados ultrapassa a quantidade de memória livre do hardware usado. No caso do Jogo da Vida, o espaço de 32.768x32.768, ocupa aproximadamente 2,35 vezes mais que a quantidade de memória livre do sistema. No caso do modelo LUCC, os espaços de 16.384x16.384 e 32.768x32.768, ocupam, respectivamente, 1,47 e 5,88 vezes mais do que a quantidade de memória livre do sistema.

Tabela 4.2 - Tempo médio de uma iteração dos modelos Jogo da Vida e LUCC.

Dimensão Espaço Celular x Tempo				
Dimensão	Jogo da Vida		LUCC	
	#GB	T(s)	#GB	T(s)
1024x1024	0.02	0.30	0.04	0.04
2048x2048	0.06	1.22	0.16	1.63
4096x4096	0.25	5.61	0.63	6.51
8192x8192	1.00	19.98	2.50	26.24
16384x16384	4.00	82.43	10.00	409.17
32768x32768	16.00	651.10	40.00	2179.69

Os gráficos das Figuras 4.3 e 4.4 mostram o tempo de processamento dos espaços celulares em função do volume de dados. A taxa de processamento do Jogo da Vida cai da ordem de 50 MB/s quando o dado é todo comportado em memória, para 25MB/s, quando o disco começa a ser usado. No modelo LUCC, esta taxa cai de aproximadamente 95MB/s para 25MB/s (16.384x16.384) e 19 MB/s (32.768x32.768).

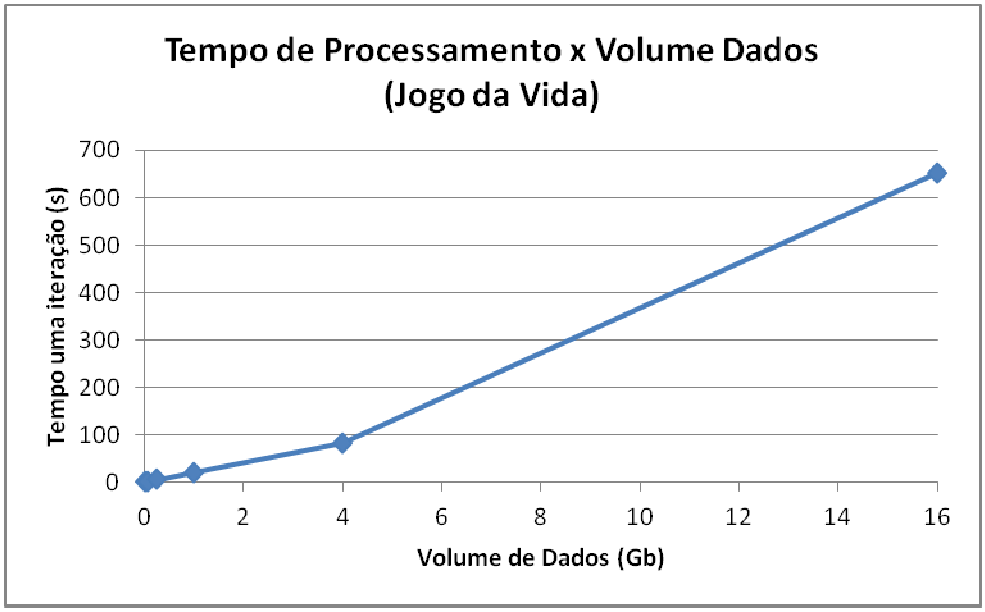


Figura 4.3 - Tempo de execução de uma iteração do modelo do Jogo da Vida.

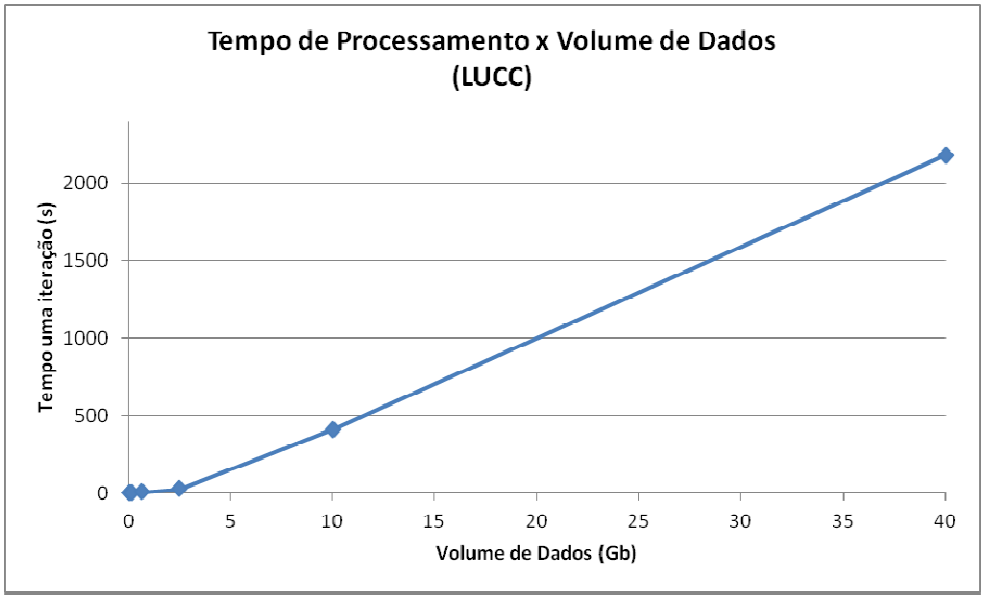


Figura 4.4 - Tempo de execução de uma iteração do modelo simplificado LUCC.

Como pode ser visto pelos resultados desta bateria, foi possível executar a simulação de uma iteração do modelo do Jogo da Vida para um espaço celular de tamanho 32.768x32.768 ocupando um volume de dados de 16GB em

aproximadamente 10 minutos. E, no caso do modelo LUCC, para um espaço do mesmo tamanho, mas com um volume de dados ainda maior, de 40 GB, uma iteração leva em torno de 36 minutos. No caso do modelo LUCC, este resultado é ainda mais importante, pois a classe de modelos representada neste teste geralmente realiza poucas iterações para construção da projeção de cenários futuros, ou seja, o tempo é bastante aceitável, ainda mais considerando um equipamento com configurações comuns.

Bateria 2: Avaliando o *Cache* do CellDB em Travessias Sequencias e Vizinhanças Isotrópicas

Para avaliar a proposta de *cache* presente no CellDB, esta bateria toma o tamanho do *cache* de células como parâmetro para o teste. O objetivo desta bateria foi verificar a influência do *cache* nas operações de travessia sequencial dos espaços celulares quando considerada uma vizinhança isotrópica de tamanho oito (vizinhança de *Moore*). O Jogo da Vida foi novamente utilizado neste experimento. Como ele faz acesso aos oito vizinhos de cada célula, o tamanho mínimo do *cache* foi escolhido de forma a comportar ao menos nove blocos de células. Este número possibilita manter em *cache* todos os blocos das células vizinhas às células do bloco em processamento, uma vez que a vizinhança é isotrópica de tamanho oito. O valor máximo de *cache* utilizado foi escolhido de forma a comportar até seis vezes o número de blocos ao longo do eixo *x* do espaço celular. Novamente, realizamos cinco execuções do jogo com três iterações.

A Tabela 4.3 apresenta o resultado deste experimento utilizando valores de *cache* proporcionais ao volume de dados manipulado no modelo (coluna *#cache*). Os tempos apresentados na coluna $T_{médio}$ representam o tempo de uma iteração do jogo da vida. A diferença entre o maior e menor tempo para cada execução dos testes é apresentada na coluna T_{r1}. A coluna T_{r2} mostra a proporção entre o pior e melhor tempo para cada tamanho de espaço celular. Como poder ser observado, o tamanho do *cache* nas travessias sequenciais

tem pouca influência no resultado final. Isto se deve, em grande parte, ao fato do sistema operacional dispor de um mecanismo de integrado ao sistema de arquivos, de forma que valores muito grandes para o *cache* de dados do CellDB têm pouco impacto no desempenho final.

Tabela 4.3 - Tamanhos usados para o cache de dados no Jogo da Vida.

Tamanho Cache						
linhas x colunas	**#cache**	**T_{min}**	**T_{max}**	**T_{r1}**	**$T_{médio}$**	**T_{r2}**
1024x1024	37.50%	0.308	0.312	1.284%	0.309	4.1%
	56.25%	0.312	0.313	0.381%	0.313	
	75.00%	0.305	0.307	0.834%	0.306	
	100.00%	0.296	0.305	3.191%	0.300	
2048x2048	14.06%	1.249	1.287	3.030%	1.261	3.2%
	18.75%	1.249	1.264	1.178%	1.254	
	37.50%	1.221	1.263	3.401%	1.231	
	56.25%	1.220	1.230	0.799%	1.223	
	75.00%	1.220	1.225	0.354%	1.222	
4096x4096	3.52%	5.718	5.809	1.595%	5.766	2.8%
	9.38%	5.714	5.835	2.108%	5.757	
	18.75%	5.604	5.662	1.035%	5.625	
	28.13%	5.584	5.803	3.923%	5.683	
	37.50%	5.589	5.689	1.785%	5.611	
8192x8192	0.88%	20.276	20.504	1.124%	20.432	2.6%
	4.69%	20.297	20.725	2.110%	20.508	
	9.38%	19.843	20.231	1.953%	20.010	
	14.06%	19.826	20.140	1.585%	19.983	
	18.75%	19.857	20.211	1.786%	20.098	
16384x16384	0.22%	81.314	90.894	11.781%	83.506	3.0%
	2.34%	81.418	90.481	11.132%	84.922	
	4.69%	80.004	88.968	11.204%	83.636	
	7.03%	79.603	86.625	8.821%	82.435	
	9.38%	79.756	89.375	12.061%	83.856	
32768x32768	0.05%	667.736	695.855	4.211%	681.297	4.6%
	1.17%	651.961	688.763	5.645%	666.697	
	2.34%	640.000	666.838	4.194%	651.951	
	3.52%	642.805	656.587	2.144%	651.103	
	4.69%	644.211	659.685	2.402%	653.108	

Este resultado é realçado pelo gráfico da Figura 4.5, em que o valor de 2,34% é capaz de comportar três linhas do espaço celular. Logo, uma vez que a memória primária não comporta todo o dado, um bom compromisso de eficiência para o *cache* é a utilização de um valor mínimo que armazene as três linhas.

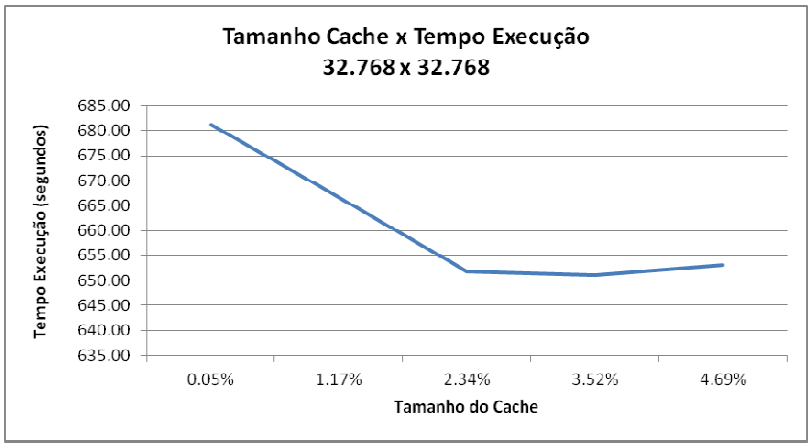


Figura 4.5 - Tamanho do cache x Tempo Execução.

Bateria 3: Avaliação das Políticas de *Cache* em Acesso Aleatório

O objetivo desta bateria foi avaliar o comportamento do CellDB diante de travessias aleatórias. Para isso, foram considerados espaços celulares de tamanhos variados, organizados em blocos com tamanho fixo de 32x32, *cache* configurado em 10% do volume dos respectivos espaços celulares e uso das políticas de *cache* FIFO e LRU. Neste experimento, tomamos o tempo médio da execução do laço mais externo do algoritmo de segregação apresentado na Seção 4.1.2, com 2.000 iterações. Para cada tamanho de espaço celular mostrado na Tabela 4.3, realizamos quinze simulações.

A Tabela 4.4 mostra o resultado desta bateria. O tempo médio apresentado é o resultado agregado das 2000 iterações, que em média resultaram no acesso aleatório a mais de 156.000 células. Como pode ser observado pelos resultados, neste tipo de situação, onde o tamanho do *cache* é muito pequeno em comparação com o volume de dados do espaço celular e realizando

acessos aleatórios às células, nenhuma das políticas clássicas de *cache* se destaca.

Tabela 4.4 - Tempo da simulação do modelo de segregação.

Política de Cache						
linhas x colunas	cache	$\#Q_{cells}$	$\#R_{cells}$	$\#F_{hits}$	$T_{média}$	$T_{diferença}$
1024x1024	FIFO	307007	159559	145782	1.14	14.93%
	LRU	307007	159559	145676	1.31	
2048x2048	FIFO	299959	154633	140587	1.32	7.12%
	LRU	299959	154633	140581	1.41	
4096x4096	FIFO	310162	161110	145373	1.82	12.15%
	LRU	310162	161110	145360	2.05	
8192x8192	FIFO	303426	155976	136181	2.00	12.06%
	LRU	303426	155976	136133	2.24	
16384x16384	FIFO	306201	158536	119761	2.03	11.43%
	LRU	306201	158536	119754	2.26	
32768x32768	FIFO	305050	156741	43696	521.23	-13.67%
	LRU	305050	156741	43684	449.98	
49152x49152	FIFO	301937	155724	0	1040.82	1.09%
	LRU	301937	155724	0	1052.16	
65536x65536	FIFO	308819	160027	0	1714.13	-2.47%
	LRU	308819	160027	0	1671.78	

Como a arquitetura do CellDB é extensível, seria possível incorporar novas políticas de *cache* mais específicas para certos tipo de modelos. No modelo em questão, poderíamos tentar utilizar um limiar de insatisfação das células nos blocos como critério para escolha de descarte, na tentativa de criar uma política mais adequada à forma como o modelo utiliza as células sorteadas, ou mesmo criar um *rank* inicial dos blocos de células para evitar sorteios desnecessários.

Outro ponto evidenciado por esta bateria é a escolha do tamanho do bloco, que tem um grande impacto no tempo de execução da simulação. No caso dos espaços celulares maiores do que 32.768x32.768, com volumes de dados maiores do que a memória primária disponível para uso, blocos maiores farão

com que seja movimentado um volume maior de dados entre o disco e a memória primária.

Bateria 4: Avaliando o caso de Vizinhanças Armazenadas em Disco

Esta bateria realiza um experimento semelhante ao da Bateria 1 com o Jogo da Vida. Ao invés de computar *on-the-fly* a vizinhança de *Moore* (vizinhança 8), as relações são recuperadas a partir de um repositório de dados mantido por um sistema NoSQL baseado em pares Chave-Valor. Assim como na Bateria 1, tomamos o tempo médio da execução do laço mais externo do algoritmo do Jogo da Vida, realizando cinco execuções do modelo com três iterações cada. O tempo médio em segundos para cada iteração desta bateria é apresentado na Tabela 4.5 na coluna T_{nosql}. Como pode ser observado, o tempo gasto usando vizinhanças armazenadas é bem superior ao de uso de uma função que computa dinamicamente esta vizinhança, conforme resultados da Bateria 1.

Tabela 4.5 - Jogo da Vida com vizinhança armazenada em sistemas NoSQL Chave-Valor.

Vizinhanças Armazenadas x *on-the-fly*				
linhas x colunas	$T_{bateria1}$ (s)	T_{nosql} (s)	T_r	$\#GB_{vizinhança}$
1024x1024	0.30	1.46	387.2%	0.05
2048x2048	1.22	6.14	402.8%	0.19
4096x4096	5.61	27.31	386.8%	0.75
8192x8192	19.98	115.65	478.7%	3.01
16384x16384	82.43	807.59	879.7%	12.02

4.2. Considerações Finais do Capítulo

Os experimentos realizados neste Capítulo mostram que o protótipo do CellDB avaliado é capaz de manipular volumes de dados muito maiores do que a quantidade de memória livre do sistema. Algumas lições podem ser aprendidas com os testes realizados. A primeira delas, é que em travessias sequenciais do

espaço celular com vizinhanças isotrópicas simples (vizinhança de Moore), como as realizadas no Jogo da Vida e no modelo simplificado LUCC, um bom compromisso de eficiência para o *cache* é a utilização de um valor mínimo que armazene pelo menos três linhas durante o processamento. Na realidade, este número deve ser compatível com o raio da vizinhança utilizada, isto é, o tamanho da janela definida para a vizinhança. Dado que o objetivo desta implementação do CellDB é prover uma ferramenta para ser usada em computadores pessoais, o uso de um *cache* com pouco consumo de memória libera o usuário para executar outros programas durante o processamento das simulações.

Para os casos onde existe uma independência de processamento entre cada uma das células de um espaço celular, como no Jogo da Vida, uma melhoria na estratégia de *cache* pode ser usada no CellDB. Na atual implementação, quando uma nova iteração começa, o CellDB precisa dos blocos do começo do espaço. Dessa forma, ele não aproveita os blocos em cache, pois estes pertencem ao final do espaço. Uma solução seria inverter a ordem de processamento a cada iteração. Isto faz com que todo o cache seja reaproveitado a cada início de nova iteração. Com isso, um crescimento no tamanho do *cache* se torna sempre justificável. Por exemplo, um cache de 90% irá jogar fora apenas 10% dos blocos de células a cada iteração com esta abordagem.

Em relação aos acessos aleatórios, trabalhados na bateria com o modelo de segregação de Schelling, se o tamanho do *cache* é muito pequeno em comparação ao volume de dados do espaço celular, nenhuma das políticas clássicas de *cache* irá se destacar. Neste caso, é preciso pensar em políticas mais específicas para tornar mais eficiente a execução deste padrão de acesso.

A bateria de testes com vizinhanças armazenadas em sistemas NoSQL da classe Chave Valor mostra que outro grande desafio é lidar com o volume de

dados das relações de vizinhança. Em alguns casos o volume de dados dessa vizinhança pode ultrapassar o volume ocupado pelos atributos do espaço celular. Por exemplo, uma vizinhança de Moore armazenada explicitamente pode ocupar o mesmo espaço que quatro atributos em ponto flutuante com 64-bit.

É preciso pensar no compromisso entre o espaço de armazenamento versus a complexidade de geração das relações. Se a vizinhança for o resultado de uma computação mais complexa, como travessias em uma rede ou consultas espaciais com predicados topológicos e métricos, deve ser feita uma comparação entre o tempo requerido para computar *on-the-fly* essas relações e o tempo de recuperação delas a partir do disco. Pelo lado do projeto do CellDB, o fato de tornar as vizinhanças extensíveis possibilita extrair o melhor de cada uma das estratégias de computação da vizinhança.

A tabela 4.6 apresenta um sumário de todos os experimentos realizados.

Tabela 4.6 - Sumário dos Experimentos.

Bateria	Conclusão	Conjectura
Grandes Espaços Celulares	desempenho linear	-
Efeito do Tamanho do Cache	Tamanho mínimo com bom compromisso	Dependente da vizinhança
Acessos Aleatórios e Políticas de Cache	Ineficiência das políticas tradicionais	Política dependente do modelo
Vizinhanças Armazenadas	ineficiente	Compromisso dependente de vizinhança

5 CONCLUSÕES E TRABALHOS FUTUROS

Esta tese apresentou uma proposta de arquitetura para acesso eficiente a dados em ferramentas de modelagem e simulação ambiental baseados em espaços celulares. Esta arquitetura se materializa no *middleware* ***CellDB,*** que fornece uma camada de acesso e gerenciamento de grandes bases de dados representadas em espaços celulares.

A arquitetura proposta e o protótipo implementado demonstram a viabilidade da hipótese apontada, de ter um *middleware* de bancos de dados capaz de desacoplar as ferramentas de modelagem de dinâmicas espacialmente explícitas, das questões de acesso e recuperação da massa de dados. Assim, ferramentas que ainda hoje são baseadas em computadores de uso pessoal podem passar a ser gerenciadas pelo *middleware,* através de uma solução de banco de dados que seja mais adequada às suas especificidades. Os resultados dos experimentos realizados com o protótipo do CellDB mostram que a arquitetura é promissora, sendo possível processar volumes de dados consideráveis em computadores pessoais de pequeno porte.

As lições aprendidas com os experimentos do Capítulo 4 mostram que outro grande desafio é lidar com o volume de dados das relações de vizinhança. Este pode ser um gargalo para a simulação de modelos, pois pode ultrapassar o volume ocupado pelos atributos do espaço celular. É preciso pensar no compromisso entre o espaço de armazenamento versus a complexidade de geração das relações. Identificamos, também, possíveis melhorias na forma de funcionamento do CellDB, como a inclusão de iteradores que possibilitem a realização de travessias que tenham a ordem de processamento invertidas a cada iteração.

Os experimentos também ajudaram a identificar necessidades de melhorias na atual arquitetura do CellDB, entre elas, aumentar a eficiência de seu subsistema de E/S de dados, que se mostrou responsável por grande parte do tempo de processamento quando se trabalha com grandes volumes de dados.

Uma possível direção futura de endereçar esta questão passa pela inclusão e avaliação de um segundo nível de *cache* na arquitetura do CellDB, que possibilite armazenar temporariamente em memória primária blocos compactados. Atualmente, existem algoritmos com taxas de compressão acima das de leitura e escrita em disco, na forma de bibliotecas de software livre, como a Google Snappy (GOOGLE, 2011), que são capazes de realizar compressões da ordem de 250 MB/s em arquiteturas de hardware como a usada nos experimentos deste trabalho (Intel CORE i-7), mas que precisam ser avaliadas em relação aos tipos de dados usados nos espaços celulares. Isto demandaria também o desenvolvimento completo do módulo de gerenciamento de entrada e saída, incluindo o trabalho de forma paralela com realização de *prefetch* de blocos compactados bem como da entrada e saída para disco.

Outra melhoria relevante aplica-se ao caso de volumes de dados comportados completamente em memória primária, onde o tempo gasto com as operações de leitura e escrita nas células, através de chamadas aos métodos *get_present*, *set_present* e *get_past*, é responsável por grande parte do tempo de processamento. Por isso, é preciso explorar o paralelismo do hardware através da programação em múltiplas threads de execução. Dado que o cache de dados é organizado em blocos e a extensibilidade dos iteradores, seria possível incluir primitivas para travessia do espaço celular a partir de múltiplas *threads* de execução.

A concepção desse *middleware* também abre a perspectiva de criar soluções alternativas para as ferramentas de modelagem, considerando cenários mais sofisticados como sua integração com tecnologias de banco de dados como o SciDB, concebido para uso em ambientes computacionais distribuídos. Uma implementação do CellDB como extensão deste SGBD, de forma embutida, possibilitaria explorar essas capacidades diretamente no servidor. Assim, todo o processamento poderia ser movido para o lado do servidor de banco de dados, evitando possíveis gargalos de comunicação entre processos. No entanto, esse tipo de estratégia depende da organização e arquitetura da

ferramenta de modelagem, bem como dos dados a serem usados nos modelos. À Figura 5.1 ilustra um possível cenário desta integração.

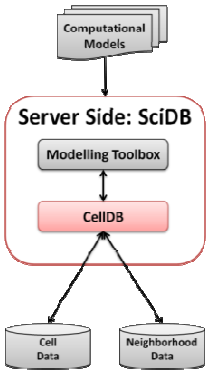


Figura 5.1 - Implementação do *middleware* CellDB embutido em um SGBD matricial.

Uma próxima etapa do projeto consiste em incluir o CellDB no TerraME, dotando este da capacidade de manipular grandes volumes de dados, bem como ampliar os tipos de fontes de dados a serem usadas com entrada e saída para seus modelos.

Outros trabalhos futuros incluem a comparação de desempenho de novas implementações do *middleware* sobre diferentes tecnologias de bancos de dados, novos experimentos com políticas de cache e melhor tratamento da vizinhança.

Para finalizar, é importante ressaltar que mesmo com todo o formalismo construído, de forma generalizada, para os sistemas relacionais ao longo de 40 anos, e com as novas propostas teóricas e conceituais produzidas nos anos 90 para as extensões espaciais e nos anos 2000 para os sistemas distribuídos de bancos de dados, ainda não existe uma única tecnologia que seja suficiente para lidar com a nova situação de múltiplas e distribuídas fontes de dados espaço-temporais e seu volume. As inovações tecnológicas de banco de dados ainda não estão completamente estabelecidas, mas passam a ter um papel fundamental na pesquisa de banco de dados com a experimentação através de protótipos e artefatos computacionais que apontam caminhos e soluções

possíveis e necessárias para lidar com questões fundamentais para o planeta em uma época onde a possibilidade de obter dados e informações é quase ilimitada.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIAR, A. P. D. **Modelagem de mudança do uso da terra na Amazônia: explorando a heterogeneidade intrarregional.** 2012. 206 p. (sid.inpe.br/MTC-m13@80/2006/08.10.18.21-TDI). Tese (Doutorado em Sensoriamento Remoto) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2006. Available from: <http://urlib.net/6qtX3pFwXQZGivnJSY/M7t7e>. Access in: 2013, Jan. 10.

ANDRADE, P. R.; MONTEIRO, A. M. V.; CÂMARA, G. Entities and Relations for Agent-Based Modelling of Complex Spatial Systems. In: Advances in Social Simulation, 2010 Second Brazilian Workshop on Social Simulation. **Proceedings...** IEEE, 2010, p. 111-118. G. P. Dimuro; A. C. R. Costa; J. S. Sichman; P. Tedesco; D. F. Adamatti; J. Balsa; L. Antunes. (Org.)

AIME, A. **GeoServer, past, present and future. 2007**. Disponível em: <http://2007.foss4g.org/presentations>. Acesso: 01 Jun. 2012.

ANDERSON, J. C.; LEHNARDT, J.; SLATER, N. **CouchDB: The Definitive Guide**. San Francisco: O'REILLY, 2010. 272p.

AVRAM, A. Gremlin, a Language for Working with Graphs. **Info Q**, Jan, 2010. Disponível em: <http://www.infoq.com/news/2010/01/Gremlin>. Acesso: 01 Jun. 2012.

BAUMANN, P.; DEHMEL, A.; FURTADO, P.; RITSCH, R.; WIDMANN, N. The multidimensional database system RasDaMan. **SIGMOD Rec.**, v. 27, n. 2, p. 575-577, Jun. 1998.

BATTY, M.; XIE, Y.; SUN, Z. Modeling urban dynamics through GIS-based cellular automata. **Computers, environment and urban systems**, v. 23, n.3, p. 205-233, 1999.

BATTY, M. **Cities and complexity:** understanding cities with cellular automata, agent-based models, and fractals. The MIT press, 2007. 592p.

BERRY, M. W.; HAZEN, B. C.; MACINTYRE, R.L.; FLAMM, R.O. LUCAS: a system for modeling land-use change. **Computational Science & Engineering**, IEEE, v. 3, n. 1, p. 24-35, 1996.

BOLLOBÁS, B. **Modern graph theory**. New York, EUA: Springer-Verlag, 1998. 408p.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. Addison-Wesley Professional, May 2005. 496p.

BROWN, P. G. Overview of sciDB: large scale array storage, processing and analysis. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2010, Indianapolis. **Proceedings...** New York, NY, USA: ACM, 2010. p. 963-968.

CÂMARA, G.; VINHAS, L.; QUEIROZ, G. R.; FERREIRA, K. R.; MONTEIRO, A. M.; CARVALHO, M.; CASANOVA, M. TerraLib: An open-source GIS library for large-scale environmental and socio-economic applications. In: HALL, B.; LEAHY, M (Eds). **Open Source Approaches in Spatial Data Handling**: advances in Geographic Information Science. Springer, 2010. p. 247-270.

CARNEIRO, T. G. S.; ANDRADE, P. R.; CÂMARA, G.; MONTEIRO, A. M. V.; PEREIRA, R. R. TerraME: an extensible toolbox for modeling nature-society interactions. **Environmental Modeling & Software**, 2012 (submetido).

CARNEIRO, T. G. S., MARETTO, R.V., CÂMARA, G. Irregular Cellular Spaces: Supporting Realistic Spatial Dynamic Modeling over Geographical Databases. In: SIMPÓSIO BRASILEIRO DE GEOINFORMÁTICA (GEOINFO), 10, 2008, Rio de Janeiro, RJ, Brasil. **Anais...** São José dos Campos: MCT/INPE, 2008.

CASANOVA, M.; CÂMARA, G.; DAVIS, C.; VINHAS, L.; QUEIROZ, G. R. **Bancos de Dados Geográficos**. Curitiba: Editora Mundo Geo, 2005. 504p.

CATTELL, R. Scalable SQL and NoSQL data stores. **ACM SIGMOD**, v. 39, n. 4, p. 12-27, December 2010.

CHANG, F.; DEAN, J.; GHEMAWAT, S.; HSIEH, W. C.; WALLACH, D. A.; BURROWS, M.; CHANDRA, T.; FIKES, A.; GRUBER, R. E. Bigtable: A distributed storage system for structured data. **ACM Transactions on Computer Systems**, v. 26, n. 4, p. 1-26, June 2008.

CHODOROW, K.; DIROLF, M. **MongoDB:** the Definitive Guide. O'REILLY, 2010. 216p.

CHOU, H-T.; DEWITT, D. J. An Evaluation of Buffer Management Strategies for Relational Database Systems. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES (VLDB '85), 11., 1985, Stockholm. **Proceedings…** Morgan Kaufmann, 1985. Volume 11, p. 127-141. Alain Pirotte and Yannis Vassiliou (Eds.).

CLEMENTINI, E.; DI FELICE, P. A Comparison of Methods for Representing Topological Relationships. **Information Sciences - Applications**, v. 3, n. 3, p. 149-178, May 1995.

COMER, D. Ubiquitous B-Tree. **ACM Computing Surveys**, v. 11, n. 2, p. 121-137, June 1979.

CROCKFORD, D. **RFC 4627:** The application/json Media Type for JavaScript Object Notation (JSON). July, 2006. Disponível em: <http://www.ietf.org/rfc/rfc4627.txt>. Acesso: 01 Jun. de 2012.

CROOKS, A. T. **The Repast Simulation/Modelling System for Geospatial Simulation**. UCL Centre for Advanced Spatial Analysis, 2007. Disponível em:

<http://www.bartlett.ucl.ac.uk/casa/publications/working-paper-123>. Acesso: Nov. 2012.

CUDRE-MAUROUX, P.; KIMURA, H.; LIM, K.-T.; ROGERS, J.; SIMAKOV, R.; SOROUSH, E.; VELIKHOV, P.; WANG, D. L.; BALAZINSKA, M.; BECLA, J.; DEWITT, D.; HEATH, B.; MAIER, D.; MADDEN, S.; PATEL, J.; STONEBRAKER, M.; ZDONIK, S. A demonstration of scidb: a science-oriented dbms. **Proc. VLDB Endow**., v. 2, n. 2, p. 1534-1537, August, 2009.

DE KONING, G. H. J.; VERBURG, P. H.; VELDKAMP, A.; FRESCO, L. O. Multi-scale modeling of land use change dynamics in Ecuador. **Agricultutal Systems**, v. 61, n. 2, p. 77-93, 1999.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, v. 51, n. 1, p. 107-113, January, 2008.

DEAN, J.; GHEMAWAT, S. **LevelDB**. Disponível em: <http://code.google.com/p/leveldb>. Acesso: 08 Ago. 2012.

DECANDIA, G.; HASTORUN, D.; JAMPANI, M.; KAKULAPATI, G.; LAKSHMAN, A.; PILCHIN, A.; SIVASUBRAMANIAN, S.; VOSSHALL, P.; VOGELS, W. Dynamo: Amazon's Highly Available Key-value Store. **ACM SIGOPS Operating System Review**, v. 41, n. 6, p. 205-220, December 2007.

DOBOS, L.; SZALAY, A.; BLAKELEY, J.; BUDAVÁRI, T.; CSABAI, I.; TOMIC, D.; MILOVANOVIC, M.; TINTOR, M.; JOVANOVIC, A. Array requirements for scientific applications and an implementation for microsoft SQL server. In: EDBT/ICDT WORKSHOP ON ARRAY DATABASE (AD'11), 2011, New York. **Proceedings…** New York: New York, 2011.

ECMA INTERNATIONAL. **ECMAScript Language Specification**. ECMA-262, 5.1 Edition, June, 2011. Disponível em: <http://www.ecma-international.org>. Acesso: 10 Jun. 2012.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of database systems**. Addison Wesley, 2006.

FAL LABS. **Kyoto Cabinet:** a straightforward implementation of DBM. Disponível em: <http://fallabs.com/kyotocabinet>. Acesso: 15 Ago. 2011.

FEITOSA, F.; LE, Q. B.; VLEK, P. L. Multi-agent simulator for urban segregation (MASUS): A tool to explore alternatives for promoting inclusive cities. **Computers, Environment and Urban Systems**, v. 35, n. 2, p. 104-115, 2011.

FEITOSA, F. F. ; LE, Q. B. ; VLEK, P. ; MONTEIRO, A. M. V. ; ROSEMBACK, R. Countering urban segregation in brazilian cities: policy-oriented explorations using agent-based simulation. **Environment & Planning B: Planning & Design**, v. 39, n. 6, p. 1131 – 1150, 2012. doi: <10.1068/b38117>.

FOLK, M.; HEBER, G.; KOZIOL, Q.; POURMAL, E.; ROBINSON, D. An overview of the HDF5 technology suite and its applications.. In: WORKSHOP ON ARRAY DATABASES (AD '11), 2011, New York. **Proceedings...** New York, NY, USA: ACM, 2011. p. 36-47. EDBT/ICDT 2011. Disponível em: <http://www.edbt.org/Proceedings/2011-Uppsala/workshops_toc.html>. Acesso: 02 Setembro 2012.

FURIERI, A. **Using SpatiaLite**. Disponível em: <http://www.gaia-gis.it/gaia-sins>. Acesso: 02 Mai. 2012.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns:** elements of reusable object-oriented software. Addison-Wesley, 1994. 416p.

GARDNER, M. Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life". **Scientific American**, n. 223, p. 120-123, 1970.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The Google file system. **ACM SIGOPS Operating Systems Review**, v. 37, n. 5, p. 29-43, October 2003.

GILBERT, N.; BANKES, S. Platforms and Methods for Agent-based Modeling. **Proceedings of the National Academy of Sciences of the USA**, v. 99, n. 3, p. 7197-7198, 2002.

GNU. **GDBM – GNU dbm**. Disponível em: <http://www.gnu.org/software/gdbm>. Acesso: 14 Ago. 2011.

GOOGLE. **Snappy, a fast compressor/decompressor**. Disponível em: <http://code.google.com/p/snappy/>. Acesso: Mar. 2011.

GOSLING, J.; JOY, B.; STEELE JR., G. L.; BRACHA, G. **The java language specification**. 3. ed. Addison-Wesley Professional, 2005.

GRAPHML. **GraphML Specification**. Disponível em: <http://graphml.graphdrawing.org/specification.html>. Acesso: Out. 2011.

GRAY, J.; LIU, D. T.; NIETO-SANTISTEBAN, M.; SZALAY, A.; DEWITT, D. J.; HEBER, G. Scientific data management in the coming decade. **ACM SIGMOD**, v. 34, n. 4, Dec. 2005.

GUTTAG, J. V.; HOROWITZ, E.; MUSSER, D. R. **Abstract Data Types and Software Validation**. Communications of the ACM, v. 21, n. 12, p. 1048-1064, Dec. 1978.

GUTTMAN, A. R-trees: a dynamic index structure for spatial search. **ACM SIGMOD**, v. 14, n. 12, p. 47-57, June 1984.

HOLT, R. C.; SCHÜRR, A.; SIM, S. E.; WINTER, A. **Graph eXchange Language**. Disponível em: <http://www.gupro.de/GXL/Introduction/intro.html>. Acesso: Nov., 2012.

HUNGER, M. Neo4j: Java-based NoSQL graph database. **Info Q**, Feb., 2010. Disponível em: <http://www.infoq.com/news/2010/02/neo4j-10>. Acesso: 09 Jun. 2012.

IERUSALIMSCHY, R.; FIGUEIREDO, L. H.; CELES, W.; Lua-an extensible extension language. **Software: Practice & Experience**, v. 26, n. 6, p. 635-652, 1996.

IBM. **IBM DB2 Spatial Extender:** user's guide and reference. Disponível em: <http://www.ibm.com.br>. Acesso: 03 Jun. 2002.

KERSTEN, M.; ZHANG, Y.; IVANOVA, M.; NES, N. SciQL, a query language for science applications. In: WORKSHOP ON ARRAY DATABASES (AD '11), 2011, Nova York. **Proceedings...** New York, NY, USA: ACM, 2011. p. 1-12.

KNIZE, N. **Why we chose MongoDB to put big data "on the map"**. Disponível em: <http://foss4g-na.org/schedule>. Acesso: 25 Jul. 2012.

KROPLA, B. **Beginning MapServer:** open source GIS Development. Apress, 2005. 448p.

LAKSHMAN, A.; MALIK, P. Cassandra: a decentralized structured storage system. **ACM SIGOPS Operating System Review**, v. 44, n. 2, p. 35-40, April 2010.

LAWDER, J. K.; KING, P. J. H. Using Space-Filling Curves for Multi-dimensional Indexing. In: BRITISH NATIONAL CONFERENCE ON DATABASES: ADVANCES IN DATABASES, 17., 2000, London. **Proceedings...** London: Springer-Verlag, 2000. p. 20-35.

LI, X.; MAGILL, W. Modeling fire spread under environmental influence using a cellular automaton approach. **Complexity International**, v. 8, 2001.

MADDEN, S. From Databases to Big Data. **Internet Computing, IEEE**, v.16, n.3, p.4-6, May-June 2012.

MARETTO, R. V. **Dynamic neighborhoods: a conceptual model and its implementation for spatial dynamics in geographic modeling**. 2011. 109 p. (sid.inpe.br/mtc-m19/2012/02.13.16.37-TDI). Dissertação (Mestrado em

Sensoriamento Remoto) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011. Disponível em: <http://urlib.net/8JMKD3MGP7W/3BC43SH>. Acesso em: 10 jan. 2013.

MELTON, J; EISENBERG, A. SQL multimedia and application packages (SQL/MM). **ACM SIGMOD**, v. 30, n. 4, p. 97-102, Dec. 2001.

MICROSOFT. **White paper:** delivering location intelligence with spatial data. Disponível em: <http://www.microsoft.com>. Acesso: 12 Dez. 2008.

MICROSOFT. **.NET Framework**. Disponível em: <http://msdn.microsoft.comy/aa308416%28v=vs.71%29.aspx>. Acesso: Nov. 2012.

NAUR, P.; RANDELL, B. **Report on a conference sponsored by the NATO SCIENCE COMMITTEE**. Garmisch, Germany, 7–11 October 1968, Brussels, Scientific Affairs Division, NATO (1969). 231p.

NEUBAUER, P. **Neo4J Spatial - GIS for the rest of us**. Disponível em: <http://www.slideshare.net/peterneubauer/2011-07oscon>. Acesso: 08 Mai. 2012.

NIEVERGELT, J.; HINTERBERGER, H.; SEVCIK, K. C. The Grid File: An Adaptable, Symmetric Multikey File Structure. **ACM Transactions on Database Systems**, v. , n. 1, p. 38-71, March 1984.

NORTH, M.J.; COLLIER, N.T.; VOS, J.R. Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. **ACM Transactions on Modeling and Computer Simulation**, v. 16, n. 1, p. 1-25, January 2006.

OBE, R.; HSU, L.; RAMSEY, P. **PostGIS in Action**. Manning Publications, 2011. 520p.

OGC. **OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1:** Common architecture. Disponível em: <http://www.opengeospatial.org>. Acesso: 17 Abr. 2012a.

OGC. **OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2:** SQL option. Disponível em: <http://www.opengeospatial.org>. Acesso: 17 Abr. 2012b.

OGC. **Web Coverage Service**. Disponível em: <http://www.opengeospatial.org>. Acesso: 17 Abr. 2012c.

OLSON, M. A.; BOSTIC, K.; SELTZER, M. Berkeley DB. In: ANNUAL CONFERENCE ON USENIX (ATEC '99), Berkeley. **Proceedings...** Berkeley, CA, USA: USENIX Association, 1999.

ORACLE. **Oracle spatial guide**. Disponível em: <http://www.oracle.com>. Acesso: 29 maio de 2003.

ORACLE. **Oracle spatial -** topology and network data models developer's guide. Disponível em: <http://www.oracle.com>. Acesso: 23 Jun. 2008.

ORACLE. **Oracle Spatial 11g GeoRaster**. Disponível em: <http://www.oracle.com>. Acesso: Nov. 2012.

PARSONS, J. A.; FONSTAD, M. A. A cellular automata model of surface water flow. **Hydrological Processes**, v. 21, n. 16, p. 2189-2195, 2007.

PCRASTER TEAM. **PCRaster documentation, Release 3.0.1**. Disponível em: <http://pcraster.geo.uu.nl/documentation/index.html>. Acesso: Nov. 2012.

PONTIUS, R. G.; CORNELL, J. D.; HALL, C. A. S. Modeling the spatial pattern of land-use change with GEOMOD2: application and validation for Costa Rica. **Agriculture, Ecosystems & Environment**, v. 85, n. 1, 2001, p. 191-203.

QUANTUM GIS DEVELOPMENT TEAM (2012). **Quantum GIS Geographic Information System**. Open Source Geospatial Foundation Project. Disponível em: <http://qgis.osgeo.org>. Acesso: 26 Mai. 2012.

RAILSBACK, S. F.; LYTINEN, S. L.; JACKSON, S. K. Agent-based Simulation Platforms: Review and Development Recommendations. **SIMULATION**, v. 82, n. 9, September 2006, p. 609-623.

RAOULT, B. **Architecture of the new MARS server**. Disponível em: <www.ecmwf.int/publications>. Acesso: 04 Jun. 2012.

RITTER, N.; RUTH, M. **GeoTIFF format specification**. Disponível em: <http://www.remotesensing.org/geotiff/spec/geotiffhome.html>. Acesso: 04 Set. 2012.

ROSENBLUETH, D. A.; GERSHENSON, C. A Model of City Traffic Based on Elementary Cellular Automata. **Complex Systems**, v. 19, p. 305-322, 2011.

SCHELLING, T. C. Dynamic models of segregation. **Journal of mathematical sociology**, Taylor & Francis, v. 1, n. 2, p. 143-186, 1971.

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.

STONEBRAKER, M.; CETINTEMEL, U. 2005. "One Size Fits All": An Idea Whose Time Has Come and Gone. In: International Conference on Data Engineering (ICDE '05), 21. **Proceedings...** IEEE Computer Society, Washington, DC, USA, 2005. p. 2-11.

STONEBRAKER, M.; MADDEN, S.; ABADI, D. J.; HARIZOPOULOS, S.; HACHEM, N.; HELLAND, P. The end of an architectural era: (it's time for a complete rewrite). In: Proceedings of the 33rd international conference on Very

large data bases (VLDB '07), 33., 2007, Vienna. **Proceedings…** Vienna: VLDB Endowment, 2007. p. 1150-1160.

STROUSTRUP, B. **The C++ programming language**. 3. ed. Addisson-Wesley, 1997. 1040p.

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS (INPE). **The design and implementation of TerraLib 5**. Disponível em: <http://www.dpi.inpe.br/terralib5/wiki>. Acesso: Nov. 2012.TOBIAS, R.; HOFMANN, C. Evaluation of free Java-libraries for social-scientific agent based simulation. **Journal of Artificial Societies and Social Simulation**, v. 7, n. 1, Jan. 2004. 6p.

TOBLER, W. R. Cellular geography. In: GALE, S.; OLSSON, G. (eds.). **Philosophy in geography**. Dordrecht, The Netherlands: D. Reidel, 1979. p. 279-386.

TURTON, I. GeoTools. In: HALL, B.; LEAHY, M (eds). **Open source approaches in spatial data handling:** advances in geographic information science. Springer, 2010. p. 153-170.

UCAR. **The netCDF file format**. Disponível em: <http://www.unidata.ucar.edu/software/netcdf/docs/netcdf_format.html>. Acesso: 04 Set. 2012.

VAN ROSSUM, G. Python for Unix/C Programmers. In: NLUUG najaarsconferentie, 1993. **Proceedings...** NLUUG, 1993. Disponível em: <http://citeseerx.ist.psu.edu>. Acesso: Nov. 2012.

VELDKAMP, A.; FRESCO, L. O. CLUE: a conceptual model to study the conversion of land use and its effects. **Ecological Modelling**, v. 85, n. 2, 1996, p. 253-270.

VERBURG, P. H.; SOEPBOER, W.; VELDKAMP, A.; LIMPIADA, R.; ESPALDON, V.; MASTURA, SS. Modeling the spatial dynamics of regional land use: the CLUE-S model. **Environmental management**, v. 30, n. 3, 2002, p. 391-405.

WADSWORTH, S. **Raster storage and processing with MongoDB**. Disponível em: <http://foss4g-na.org/schedule>. Acesso: 25 Jul. 2012.

WARMERDAM, F. The geospatial data abstraction library. In: HALL, B.; LEAHY, M. (eds). **Open source approaches in spatial data handling**: advances in geographic information science. Berlin: Springer, 2010. p. 87-104.

WORLD METEOROLOGICA ORGANIZATION (WMO). **GRIB Specification**. Disponível em: <http://www.wmo.int/pages/prog/www/WDM/Guides/Guide-binary-2.html>. Acesso: Set. 2012.

10GEN. **MongoDB Manual:** 2d Geospatial Indexes. Disponível em: <http://docs.mongodb.org/manual/core/geospatial-indexes>. Acesso: 23 Jan. 2013.

APÊNDICE A - ESPECIFICAÇÃO DO CATÁLOGO INTERNO DE METADADOS DO CellDB

Este apêndice contém uma especificação, na forma de esquemas e documentos XML, dos metadados mantidos internamente pelo CellDB para acesso e armazenamento dos espaços celulares e vizinhanças. Esses esquemas foram implementados dentro do ambiente da TerraLib 5.0, compartilhando as definições de seu módulo de acesso a dados.

A.1 Espaço Celular

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:te_common="http://www.terralib.org/common"
            xmlns:te_cs="http://www.terralib.org/cellspace"
            xmlns:te_rst="http://www.terralib.org/raster"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.terralib.org/cellspace"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            xml:lang="en">


  <xsd:import namespace="http://www.terralib.org/common"
              schemaLocation="../common/common.xsd"/>
  <xsd:import namespace="http://www.terralib.org/raster"
              schemaLocation="../raster/raster.xsd"/>

  <xsd:element name="CellularSpaceList" type="te_cs:CellularSpaceListType" />

  <xsd:complexType name="CellularSpaceListType">
    <xsd:sequence>
      <xsd:element name="CellularSpace" type="te_cs:CellularSpaceType"
                   minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="version" type="xsd:string" use="required"/>
    <xsd:attribute name="release" type="xsd:date" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="CellularSpaceType">
    <xsd:annotation>
      <xsd:documentation>Cellular Space Definition</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="Mask" type="te_cs:MaskType" minOccurs="0" maxOccurs="1"
/>
      <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Title" type="xsd:string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Description" type="xsd:string" minOccurs="1"
maxOccurs="1" />
      <xsd:element name="Grid" type="te_rst:GridType" minOccurs="1" maxOccurs="1"
/>
      <xsd:element name="AttributeList" type="te_cs:AttributeListType"
```

```
minOccurs="1" maxOccurs="1" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required" />
    <xsd:attribute name="type" type="xsd:string" use="required" />
  </xsd:complexType>

  <xsd:complexType name="AttributeListType">
    <xsd:annotation>
      <xsd:documentation>A simple list of attributes for a cellular
space</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="Attribute" type="te_cs:AttributeType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="AttributeType">
    <xsd:annotation>
      <xsd:documentation>Describes the source of data for an attribute of a
cellular space</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="DataSourceId" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="DataSetName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="PropertyId" type="xsd:nonNegativeInteger" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="idx" type="xsd:nonNegativeInteger" use="required" />
  </xsd:complexType>

  <xsd:complexType name="MaskType">
    <xsd:annotation>
      <xsd:documentation>Describes the source of data for a given cell space
mask</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="DataSourceId" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="DataSetName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
      <xsd:element name="PropertyName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

A.2 Vizinhança

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:te_cs="http://www.terralib.org/cellspace"
            xmlns:te_dt="http://www.terralib.org/datatype"
```

```
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.terralib.org/cellspace"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            xml:lang="en">

  <xsd:annotation>
    <xsd:appinfo>Cellular Space Schema</xsd:appinfo>
    <xsd:documentation>Definition of a cellular space</xsd:documentation>
  </xsd:annotation>

  <xsd:import namespace="http://www.terralib.org/datatype"
schemaLocation="../datatype/datatype.xsd"/>

  <xsd:element name="NeighborhoodList" type="te_cs:NeighborhoodListType">
    <xsd:annotation>
      <xsd:documentation>This is the root element for a cellular space
list</xsd:documentation>
    </xsd:annotation>
  </xsd:element>

  <xsd:complexType name="NeighborhoodListType">
    <xsd:sequence>
      <xsd:element name="Neighborhood" type="te_cs:NeighborhoodType"
minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="version" type="xsd:string" use="required"/>
    <xsd:attribute name="release" type="xsd:date" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="NeighborhoodType">
    <xsd:annotation>
      <xsd:documentation>Neighborhood Definition</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Title" type="xsd:string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Description" type="xsd:string" minOccurs="1"
maxOccurs="1" />
      <xsd:element name="DataSourceId" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required" />
    <xsd:attribute name="type" type="te_cs:NeighborhoodTypeType" use="required"
/>
  </xsd:complexType>

  <xsd:simpleType name="NeighborhoodTypeType">
    <xsd:annotation>
      <xsd:documentation>The type of a neighborhood</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Simple" />
      <xsd:enumeration value="Weighted" />
      <xsd:enumeration value="Bipartite" />
      <xsd:enumeration value="WeightedBipartite" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

A.3 Fontes de Dados

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:te_common="http://www.terralib.org/common"
            xmlns:te_da="http://www.terralib.org/dataaccess"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.terralib.org/dataaccess"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            xml:lang="en">

  <xsd:import namespace="http://www.terralib.org/common"
              schemaLocation="../common/common.xsd"/>

  <xsd:element name="DataSource" type="te_da:DataSourceType" />

  <xsd:complexType name="DataSourceType">
    <xsd:annotation>
      <xsd:documentation>Data Source Instance</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Description" type="xsd:string"
                   minOccurs="0" maxOccurs="1" />
      <xsd:element name="ConnectionInfo" type="te_common:ParameterListType"
                    minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" />
    <xsd:attribute name="type" type="xsd:string" use="required"/>
    <xsd:attribute name="access_driver" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:element name="DataSourceList" type="te_da:DataSourceListType" />

  <xsd:complexType name="DataSourceListType">
    <xsd:annotation>
      <xsd:documentation>A list of data sources</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="te_da:DataSource" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="version" type="xsd:string" use="required"/>
    <xsd:attribute name="release" type="xsd:date" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="DataSourceTypeType">
    <xsd:annotation>
      <xsd:documentation>The type of data source</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="Title" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="Description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
```

```
  </xsd:complexType>

</xsd:schema>
```

A.4 Tipos Elementares

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:te_common="http://www.terralib.org/common"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.terralib.org/common"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            xml:lang="en">

  <xsd:annotation>
    <xsd:appinfo>General declarations for TerraLib schemas</xsd:appinfo>
    <xsd:documentation>This schema contains the basic set of elements and data
types used in other TerraLib schema documents</xsd:documentation>
  </xsd:annotation>

  <xsd:import namespace="http://www.w3.org/1999/xlink"
              schemaLocation="../../ogc/xlink/1.0.0/xlinks.xsd"/>

  <xsd:complexType name="OnlineResourceType">
    <xsd:attributeGroup ref="xlink:simpleLink"/>
  </xsd:complexType>

  <xsd:complexType name="ParameterListType">
    <xsd:sequence>
      <xsd:element name="Parameter" type="te_common:ParameterType"
                   minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="ParameterType">
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="Value" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

APÊNDICE B - IMPLEMENTAÇÃO MODELOS NO CellDB

Este apêndice contém ilustrações do código fonte C++ usado nos testes apresentados no Capítulo 4.

B.1 Jogo da Vida com CellDB

O Quadro B.1 mostra o trecho de código da primeira parte da implementação deste modelo, com a API do CellDB, onde é realizada a criação do espaço celular e a configuração dos estados de cada célula. No exemplo, o espaço celular possui um único atributo numérico de 8-bits e uma dimensão de 1024 linhas por 1024 colunas.

```
01  // estados das células
02  #define VIVO   1
03  #define MORTO  2
04
05  // índice do atributo com o estado da célula
06  #define ESTADO  0
07
08  boost::random::uniform_real_distribution<> dist(0.0, 1.0);
09
10  bool set_estado_inicial(cell& c)
11  {
12    double v = dist(random_gen);
13
14    if(v > 0.5)
15      c.set_present(ESTADO, VIVO);
16    else
17      c.set_present(ESTADO, MORTO);
18
19    return true;
20  }
21
22  cell_space* cria_espaco()
23  {
24   cell_space_attribute_def atributo("estado", UINT8, 0.0);
25
26   cell_space_def cs_def("jogo-da-vida", 1024, 1024, atributo);
27
28   cell_space* cs = celldb::create(cs_def);
29
30   cell_space_iterator* it = cs->get_iterator();
31
32   while(it->next())
33     set_estado_inicial(it->get());
34
35   return cs;
36  }
```

Quadro B.1 - Criação de um espaço celular para o jogo da vida.

A implementação completa do Jogo da Vida é mostrada no trecho de código do Quadro B.2, onde é possível observar:

a) preparação do espaço celular contendo as células do jogo da vida (linha 28);

b) a definição de uma vizinhança baseada em uma função que computa on-the-fly os oito vizinhos de cada célula (linha 30);

c) o uso de um iterador para percorrer sequencialmente as células do espaço celular (linha 36);

d) leitura e escrita de atributos através dos métodos get_past (linha 18) e set_present (linhas 21 e 23);

e) a sincronização dos estados presente e passado dos atributos das células a cada iteração do jogo (linha 36).

```
01  int contador = 0;
02
03  int numero_iteracoes = 300;
04
05  void incrementa_se_estado_vivo(cell& c, cell& n)
06  {
07    if(n.get_past(ESTADO) == VIVO)
08      ++count;
09  }
10
11  void joga(cell& c)
12  {
13    contador = 0;
14
15    neighborhood* nb = neighborhood_manager::get(c);
16
17    nb->apply(c, incrementa_se_estado_vivo);
18
19    double estado = c.get_past(ESTADO);
20
21    if((state == VIVO) && ((contador > 3) || (contador < 2)))
22      c.set_present(ESTADO, VIVO);
23    else if((estado == MORTO) && (contador == 3))
24      c.set_present(ESTADO, VIVO);
25  }
26
27  void jogo_da_vida()
28  {
29    cell_space* cs = celldb::prepare("jogo-da-vida");
30
31    neighborhood* n = new moore(cs);
32
33    for(int i = 0; i != numero_iteracoes; ++i)
34    {
35      cs->sync();
36
37      cell_space_iterator* it = cs->get_iterator();
38
39      while(it->next())
40        joga(it->get());
41    }
42  }
```

Quadro B.2 - Jogo da Vida

B.2 Modelo de Segregação com CellDB

O trecho de código mostrado no Quadro B.3 apresenta a primeira parte da implementação deste modelo, onde é realizada a criação do espaço celular e a configuração dos estados de cada célula. O espaço celular criado possui um único atributo numérico em ponto flutuante de 64-bit e uma dimensão de 4096 linhas por 4096 colunas.

```
01  #define LIMIAR_SATISFACAO 4
02
03  #define TAXA_DESOCUPACAO 0.05
04
05  // estados/cores das células
06  #define BRANCA   0.0
07  #define VERMELHA 1.0
08  #define PRETA    2.0
09
10  boost::random::uniform_real_distribution<> dist(0.0, 1.0);
11
12  void set_estado_inicial(cell& c)
13  {
14    double rvalue = dist(random_gen);
15
16    if(rvalue < TAXA_DESOCUPACAO)
17      c.set_present("cor", BRANCA);
18    else if(rvalue < ((1.0 + TAXA_DESOCUPACAO) / 2.0))
19      c.set_present("cor", VERMELHA);
20    else
21      c.set_present("cor", PRETA);
22  }
23
24  cell_space* cria_espaco()
25  {
26   cell_space_attribute_def atributo("cor", DOUBLE, 0.0);
27
28   cell_space_def cs_def("segregacao", 4096, 4096, atributo);
29
30   cell_space* cs = celldb::create(cs_def);
31
32   cell_space_iterator* it = cs->get_iterator();
33
34   while(it->next())
35     set_estado_inicial(it->get());
36
37   return cs;
38  }
```

Quadro B.3 - Criando um espaço celular para o modelo de segregação de Schelling

A implementação completa do modelo é mostrada no trecho de código do Quadro B.4, onde é possível observar:

a) a preparação do espaço celular contendo as células do modelo, considerando apenas um estado para os atributos das células (parâmetro *false* na linha 84);

b) a definição de uma vizinhança baseada em uma função que computa *on-the-fly* os oito vizinhos de cada célula (linha 86);

c) o uso de um *iterador* para percorrer as células do espaço celular de forma aleatória (linha 88);

d) leitura e escrita de atributos através dos métodos *get_present* e *set_present* usando o nome do atributo ao invés de um índice numérico (linhas 9, 15, 75 e 77);

```
01  int contador;
02
03  int numero_iteracoes = 300;
04
05  double cor_global;
06
07  void conta(cell& c, cell& celula_vizinha)
08  {
09    if(celula_vizinha.get_present("cor") == cor_global)
10      ++contador;
11  };
12
13  void esta_infeliz(double cor, cell& c)
14  {
15    if(c.get_present("cor") != cor)
16      return false;
17
18    contador = 0;
19
20    cor_global = cor;
21
22    neighborhood* nb = neighborhood_manager::get(c);
23
24    nb->apply(c, conta);
25
26    return (contador < LIMIAR_SATISFACAO);
27  }
28
29  void esta_disponivel(double cor, cell& c)
30  {
31    if(c.get_present("cor") != BRANCA)
32      return false;
33
34    contador = 0;
35
36    cor_global = cor;
37
38    neighborhood* nb = neighborhood_manager::get(c);
39
40    nb->apply(c, conta);
41
42    return (contador >= LIMIAR_SATISFACAO);
43  }
44
```

```
45  template<class F> inline cell get_rand_cell(cell_space_iterator* it,
46                                              double cor,
47                                              F condicao)
48  {
49    while(it->next())
50    {
51      cell c = it->get();
52
53      if(condicao (cor, c))
54        return c;
55    }
56
57    return cell();
58  };
59
60  bool partida(cell_space_iterator* it, double cor)
61  {
62    cell celula_antiga = get_rand_cell(it, cor, esta_infeliz);
63
64    if(!celula_antiga.is_valid())
65      return false;
66
67    cell celula_nova = get_rand_cell(it, cor, esta_disponivel());
68
69    if(!celula_nova.is_valid())
70      return false;
71
72  // troca celulas
73    double celula_antiga_v = celula_antiga.get_present("cor");
74
75    celula_antiga.set_present("cor", celula_nova.get_present("cor"));
76
77    celula_nova.set_present("cor", celula_antiga_v);
78
79    return true;
80  }
81
82  void schelling()
83  {
84    cell_space* cs = celldb::prepare("segregacao", false);
85
86    neighborhood* n = new moore(cs);
87
88    cell_space_iterator* it = cs->get_random_iterator();
89
90    for(uint32_t i = 0; i != numero_iteracoes; ++i)
91    {
92      bool cr = partida(it.get(), VERMELHA);
93      bool cb = partida (it.get(), PRETA);
94
95      if(!(cr || cb))
96        break;
97    }
98  }
```

Quadro B.4 - Modelo de Schelling

B.5 Modelo LUCC

Na implementação mostrada no trecho de código do Quadro B.5 podemos observar:

a) a preparação do espaço celular contendo atributos que são somente de leitura e atributos de leitura e escrita (linhas 69 a 77);

b) a definição de uma vizinhança baseada em uma função que computa *on-the-fly* os oito vizinhos de cada célula (linha 79);

c) o uso de *iteradores* para percorrer sequencialmente as células durante as etapas que determinam o potencial de desflorestamento das células e a alocação do desflorestamento (linhas 85 e 94);

```
001 #define ALOCACAO 0.01        // demanda anual de alocação
002
003 // atributos de leitura e escrita
004 #define POTENCIAL                0  // potencial de desflorestamento da célula
005 #define DESFLORESTAMENTO         1  // quantidade de desflorestamento na célula
006
007 // atributos somente de leitura
008 #define DISTANCIA_AREAS_URBANAS 2 // distância a áreas urbanas
009 #define CONEXAO_MERCADOS        3 // conexão a mercados
010 #define AREA_PROTECAO           4 // porcentagem da célula que deve ser preservada
011
012 // variáveis globais auxiliares
013 int contador_vizinhos;
014 double desflorestamento_vizinhos;
015 double total_potencial;
016 double total_demanda;
017 double porcentagem_potencial_desflorestamento;
018
019 void soma_desfloretamento_vizinhos(cell& c, cell& n)
020 {
021   desflorestamento_vizinhos += n.get_present(DESFLORESTAMENTO);
022
023   ++contador_vizinhos;
024 }
025
026 void calcula_potencial(cell& c)
027 {
028   c.set_present(POTENCIAL, 0.0);
029
030   contador_vizinhos = 0;
031
032   desflorestamento_vizinhos = 0.0;
033
034   neighborhood* nb = neighborhood_manager::get(c);
035
036   nb->apply(c, soma_desfloretamento_vizinhos);
037
038   double desflorestamento = c.get_present(DESFLORESTAMENTO);
039
040   double media = desflorestamento_vizinhos /
041 static_cast<double>(contador_vizinhos);
042
043   double esperado = 0.7300 * media
044                   - 0.1500 * std::log10(c.get_present(DISTANCIA_AREAS_URBANAS))
045                   + 0.0500 * c.get_present(CONEXAO_MERCADOS)
046                   - 0.0700 * c.get_present(AREA_PROTECAO)
047                   + 0.7734;
048
049   if(esperado > (1.0 - desflorestamento))
050     esperado = (1.0 - desflorestamento);
051
052   double val = esperado - desflorestamento;
053
054   c.set_present(POTENCIAL, val);
055
056   total_potencial += val;
057 }
058
```

```
059 void desfloresta(cell& c)
060 {
061   double nova_area = c.get_present(POTENCIAL) *
062 porcentagem_potencial_desflorestamento;
063
064   double val = c.get_present(DESFLORESTAMENTO) + nova_area;
065
066   c.set_present(DESFLORESTAMENTO, val);
067 }
068
069 void lucc()
070 {
071   std::vector<cell_space_attribute> atributos;
072
073   atributos.push_back("potencial", READ_WRITE);
074   atributos.push_back("desflorestamento", READ_WRITE);
075   atributos.push_back("distancia_areas_urbanas", READ_ONLY);
076   atributos.push_back("conexao_mercados", READ_ONLY);
077   atributos.push_back("area_protecao", READ_ONLY);
078
079   cell_space* cs = celldb::prepare("lucc", atributos));
080
081   neighborhood* n = new moore(cs);
082
083   for(int i = 0; i != numero_iteracoes; ++i)
084   {
085     total_potencial = 0.0;
086
087     cell_space_iterator* it = cs->get_iterator();
088
089     while(it->next())
090       calcula_potencial(it->get());
091
092     total_demanda = ALOCACAO * cs->get_num_cells();
093
094     porcentagem_potencial_desflorestamento = total_demanda / total_potencial;
095
096     it = cs->get_iterator();
097
098     while(it->next())
099       desfloresta(it->get());
100   }
101 }
102
```

Quadro B.5 - Modelo de Schelling