



MINISTÉRIO DA CIÊNCIA E TECNOLOGIA

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

ENGENHARIA DE *SOFTWARE* APLICADA À
CONSTRUÇÃO DE MODELOS ESPACIAIS DINÂMICOS

Tiago Garcia de Senna Carneiro

Monografia de Qualificação em Computação Aplicada, orientada pelo
Dr. Gilberto Câmara

INPE
São José dos Campos

2003

RESUMO

O modelo mais aceito e difundido para representação matemática da realidade geográfica considera duas abstrações básicas: os conceitos de campos e objetos. Infelizmente, esse modelo lida somente com uma realidade estática e invariável, eles não descrevem mudanças espaciais e não lidam com objetos complexos que podem consistir de partes que interagem, ou apresentar variações em diferentes escala. A maioria das pesquisas em modelagem espacial dinâmica resultou em modelos específicos para um determinado domínio construídos sobre soluções de *software* proprietárias. Uma ferramenta de modelagem de domínio público que possa ser utilizada para a construção de modelos intercambiáveis e de fácil integração encontraria respaldo na comunidade de modeladores. Por estas razões, esse trabalho procura identificar os principais requisitos de uma ferramenta computacional de suporte à construção de modelos dinâmicos espaciais em diferentes domínios de conhecimento e identificar as técnicas de engenharia de *software* adequadas à construção de uma ferramenta nesses moldes.

SUMÁRIO

CAPÍTULO 1	1
INTRODUÇÃO	1
1.1 <i>A demanda por modelos dinâmicos espaciais no Brasil e no Mundo</i>	1
1.2 <i>Motivação e Objetivo do Trabalho</i>	3
1.3 <i>Organização do texto</i>	4
CAPÍTULO 2	6
PROJETO E DESENVOLVIMENTO DE <i>SOFTWARE</i> REUTILIZÁVEL	6
2.1 <i>Introdução</i>	6
2.2 <i>Requisitos de software reutilizável</i>	6
2.3 <i>Projetos de Software Reutilizável</i>	8
2.3.1 <i>Bibliotecas de Software (Toolkit)</i>	8
2.3.2 <i>Arcabouços de Software (Framework)</i>	9
2.3.3 <i>Ambientes de Programação</i>	10
2.3.4 <i>Discussão</i>	11
2.4 <i>Reuso de Código na Programação Orientada a Objetos</i>	11
2.4.1 <i>Os conceitos de tipo e classe</i>	12
2.4.2 <i>Reuso de código por meio de herança e composição de objetos</i>	12
2.5 <i>O uso de padrões para aumentar a reusabilidade de um projeto de software</i>	15
2.6 <i>Contribuições da programação genérica para a reusabilidade de um projeto de software</i>	17
CAPÍTULO 3	21
MODELOS DINÂMICOS ESPACIAIS	21
3.1 <i>Introdução</i>	21
3.1.1 <i>O paradigma dos quatro universos</i>	22
3.2 <i>Espaço</i>	23
3.3 <i>Tempo</i>	25
3.4 <i>Comportamento</i>	28
3.5 <i>Modelos Espaço-Temporais</i>	34
3.5.1 <i>Autômatos Celulares</i>	36
CAPÍTULO 4	39
REQUISITOS DE FERRAMENTAS COMPUTACIONAIS PARA MODELAGEM ESPACIAL DINÂMICA	39
4.1 <i>Demandas da atividade de modelagem</i>	39
4.2 <i>Demandas da ferramenta de modelagem</i>	42
4.3 <i>Considerações finais</i>	43

CAPÍTULO 1

Introdução

1.1 A demanda por modelos dinâmicos espaciais no Brasil e no Mundo

Nas últimas décadas, a busca por modelos que permitam avaliações integradas e que combinem fatores econômicos, ecológicos, demográficos e climáticos tem sido um compromisso da comunidade científica nacional e internacional. Embora sejam principalmente os impactos negativos que motivem esse interesse, nem sempre as mudanças são negativas.

No Brasil, o processo de implementação de políticas públicas ainda é baseado em abordagens estritamente setoriais, nas quais cada setor do Governo adota decisões com percepção restrita de seu impacto. Após a conferência RIO 92, numa tentativa de estabelecer um processo de decisão gerencial integrada, o governo brasileiro comprometeu-se a adotar os princípios do *desenvolvimento sustentável*. Contudo, como este conceito ainda não tem uma formulação científica sólida, as conseqüências práticas desta decisão foram limitadas. No ano de 2002, um grupo de instituições brasileiras, entre elas INPA – Instituto Nacional de Pesquisas da Amazônia, INPE – Instituto Nacional de Pesquisas Espaciais, IDSM – Instituto de Desenvolvimento Sustentável Mamirauá, LNCC – Laboratório Nacional de Computação Científica, IMPA – Instituto de Matemática Pura e Aplicada e MPEG – Museu Paraense Emílio Goeldi, uniu-se para estudar, desenvolver, comparar, integrar e diferenciar os múltiplos aspectos do conceito de sustentabilidade em diferentes condições sócio-ambientais e propôs a *Rede Temática de Pesquisa em Modelagem da Amazônia*. A preocupação se volta, principalmente, à região da Amazônia Legal, devido ao acelerado processo de ocupação do território nas últimas décadas, que ocasionou um aumento na área desflorestada da Amazônia Legal brasileira de 10 milhões

de hectares em 1970 para aproximadamente 59 milhões de hectares em 2000 (Alves 2002), correspondendo a 14% da floresta original.

Um dos objetivos da Rede Temática de Pesquisa em Modelagem da Amazônia é desenvolver modelos computacionais capazes de auxiliar no prognóstico da dinâmica espaço-temporal dos sistemas ecológicos e sócio-econômicos em diferentes escalas geográficas, auxiliando no processo de tomada de decisão nos níveis local, regional e nacional, ao fornecer ferramentas de simulação e modelagem para a construção de cenários.

Como meta para os seus três primeiros anos, a rede deverá desenvolver modelos integrados multi-escala que incorporem diferentes dimensões da sustentabilidade na Amazônia, entre elas a dinâmica populacional, a biodiversidade, as mudanças de uso do solo e os condicionantes climáticos e hidrológicos, e produzir instrumentos computacionais que permitam observar estes modelos.

Em um panorama internacional, o projeto *Land Use and Cover Change (LUCC)*, um elemento dos dois principais programas de pesquisas sobre mudanças globais, o *International Geosphere-Biosphere Programme (IGBP)* e o *International Human Dimensions Programme on Global Environmental Change (IHDP)*, é um dos principais esforços na busca por modelos dinâmicos espaciais que permitam o estudo de sistemas biofísicos e sociais integrados. O projeto LUCC tem como incumbência prover informações sobre as mudanças passadas na cobertura do solo da Terra, ajudar o desenvolvimento de projeções da futura dinâmica de uso e cobertura do solo e identificar regiões críticas que são particularmente vulneráveis a mudanças ambientais globais. O principal objetivo do projeto LUCC é melhorar o entendimento e adquirir novos conhecimentos sobre mudanças regionais e interativas entre usos e cobertura da terra (Turner et al. 1995).

1.2 Motivação e Objetivo do Trabalho

O modelo mais aceito e difundido para representação matemática da realidade geográfica considera duas abstrações básicas: os conceitos de **campos** e **objetos**. O modelo de objetos representa a realidade como uma superfície ocupada por objetos discretos e identificáveis, os quais podem possuir atributos espaciais e descritivos próprios. O modelo de campos representa a realidade como uma superfície contínua, sobre a qual os fenômenos geográficos a serem observados variam segundo diferentes distribuições (Goodchild 1992).

Estes dois conceitos são utilizados pela maioria dos sistemas de informação geográfica (SIG) atuais. Na sua forma mais simples eles lidam somente com uma realidade estática e invariável, eles não descrevem mudanças espaciais e não lidam com objetos complexos que podem consistir de partes que interagem, ou apresentar variações em diferentes escalas (Burrough e Frank 1995).

Formas simples de análise dinâmica têm sido disponibilizadas em alguns SIGs matriciais na forma de álgebra de mapas (Tomlin 1990). Abordagens genéricas com respeito à modelagem dinâmica em GIS têm sido desenvolvidas (van Deursen e Kwadijk 1993 citado em Burrough e Frank 1995). Entretanto, até recentemente, as ferramentas de modelagem dinâmica não estão completamente integradas a um SIG, que é utilizado somente para armazenamento e visualização dos dados geográficos. Esta limitação geralmente implica em muito esforço para a conversão dos dados e em problemas de redundância e inconsistência dos mesmos. As ferramentas de modelagem também não apresentam as capacidades de análise presentes em um SIG, conseqüentemente, a habilidade dessas ferramentas de trabalhar com relações espaciais é limitada.

A maioria das pesquisas em modelagem espacial dinâmica resultou em modelos específicos para um determinado domínio construídos sobre soluções de *software* proprietárias (Veldkamp e Fresco 1996; Verburg et al. 2002; Soares-Filho 2002; White et al. 1997; Lim et al. 2002). Isto dificulta a comparação dos resultados obtidos pelas diferentes pesquisas e,

geralmente, impede que modelos diferentes possam ser integrados em um único modelo ou até mesmo compartilhados entre os pesquisadores. Por isso, uma ferramenta de modelagem de domínio público que possa ser utilizada para a construção de modelos intercambiáveis e de fácil integração encontraria respaldo na comunidade de modeladores (Parker et al. 2001).

Diante deste cenário, esse trabalho tem como objetivos a identificação dos principais requisitos de uma ferramenta computacional de suporte à construção de modelos dinâmicos espaciais em diferentes domínios de conhecimento e identificação das técnicas de engenharia de *software* adequadas à construção de uma ferramenta nesses moldes.

Para tal, o problema da construção de modelos computacionais será abordado do ponto de vista da reusabilidade de *software*. Acredita-se que os desafios encontrados no projeto e no desenvolvimento de uma ferramenta para a construção de modelos espaciais dinâmicos surgem, fundamentalmente, de problemas envolvidos na elaboração de uma ferramenta que constrói outras ferramentas computacionais. Em última instância, trata-se de um código que precisa ser reutilizado de alguma maneira para a construção de um novo código. Por esta razão, o projeto de uma ferramenta como a que se deseja precisa favorecer a reusabilidade do seu código.

Quanto maior a reusabilidade de um código, mais facilmente ele pode ser utilizado por seu usuário e maior a classe de soluções que podem ser construídas através da sua reutilização. Da mesma maneira, uma ferramenta de modelagem de alta reusabilidade, facilmente, seria utilizada na construção de modelos e também seria adequada à construção de uma ampla gama de modelos.

1.3 Organização do texto

O segundo capítulo do texto apresenta a fundamentação teórica necessária para o projeto e desenvolvimento de *softwares* reutilizáveis. Questões trazidas pela orientação a objetos,

padrões de projeto e programação genérica são consideradas. No terceiro capítulo, é dada uma visão geral sobre os modelos básicos necessários para a construção de modelos espaço-temporais segundo a demanda colocada pela *Rede Temática de Pesquisa em Modelagem da Amazônia*. O capítulo 4 procura identificar os requisitos de uma ferramenta destinada à construção de modelos computacionais de fenômenos espaço-temporais.

CAPÍTULO 2

Projeto e Desenvolvimento de *Software* Reutilizável

2.1 Introdução

As decisões envolvidas no projeto e no desenvolvimento de uma ferramenta para a construção de modelos computacionais tratam, fundamentalmente, de problemas envolvidos na elaboração de uma ferramenta que constrói outros sistemas computacionais. Em última instância, trata-se de um código que pode ser reutilizado para a construção de diferentes produtos, que possuem alguns conceitos em comum. Ao se conceber uma ferramenta que seja capaz de construir modelos espaço-temporais de diferentes domínios, é preciso favorecer a reusabilidade da arquitetura conceitual de seu código. A ferramenta de modelagem que, mais facilmente e freqüentemente, puder ser reutilizada para a construção de modelos é, em tese, a melhor ferramenta.

2.2 Requisitos de *software* reutilizável

Softwares cujos projetos visam principalmente a reutilização de suas funcionalidades em outros projetos de *software*, a partir daqui denominados *softwares reutilizáveis*, devem ser expressivos e legíveis para garantir eficiência na sua reutilização, manutenção e evolução; devem ser extensíveis para que sejam adequados à construção de soluções para a maior gama de problemas possíveis; e devem ser flexíveis para que possam ser corrigidos ou melhorados sem que essas alterações impliquem em reescrita dos *softwares* que o reutilizam ou de seus componentes.

- **Expressividade:** A expressividade é uma métrica utilizada para mensurar a facilidade com que uma linguagem de programação ou um código pode ser utilizado para expressar a solução de um problema. Quanto mais expressivo é um código,

mais eficiente é sua reutilização para a construção de um novo código. A expressividade de um código ou de uma linguagem está diretamente relacionada ao vocabulário escolhido para expressar os conceitos, isto é, tipos e objetos, com os quais trabalha e as operações definidas sobre esses conceitos.

- **Legibilidade:** Quanto mais legível é um código, melhor documentada fica a solução nele implementada e mais fácil é sua manutenção e evolução. Desta maneira, um código legível tem maior chances de ser reutilizado para a construção de diferentes soluções.
- **Extensibilidade:** Para que um código possa ser utilizado na construção de uma vasta gama de soluções, ele deve possuir mecanismos para que seu usuário construa novos tipos a partir dos tipos que ele predefine. Somente desta maneira, os conceitos do usuário poderão ser expressos pela composição de conceitos conhecidos.
- **Flexibilidade:** Um projeto de *software* é dito flexível se parte do seu código puder sofrer pequenas modificações sem que isso implique em alterações massivas e encadeadas em outras partes do seu código. A maneira fácil de se conseguir flexibilidade para um componente de software é separando sua implementação da sua interface. O fraco acoplamento entre estas duas partes permite que a implementação do componente seja alterada, possivelmente em tempo de execução, sem que seus clientes percebam.

Um *software* reutilizável deve ter a interface de todos seus componentes separada da implementação, para permitir que implementações possam ser alteradas sem que as aplicações que o reutilizam e que outros de seus componentes precisem ser modificados.

Todo cuidado deve ser tomado com a especificação da interface, porque ela precisa conhecer conceitos específicos do domínio do problema para ser expressiva e precisa ser

simples o suficiente para ser legível; e porque ela precisa ser extensível e geral o suficiente para poder ser utilizada na construção de várias soluções e, ao mesmo tempo, deve se manter estável ao longo do tempo, isto é, não deve sofrer alterações grandes ou frequentes. Mudanças na interface de um *software*, geralmente, implicam que na rescrita de todos os seus clientes.

A estabilidade da interface de um *software* depende da correta identificação dos tipos e objetos que ela define e dos relacionamentos entre estes tipos e objetos, ou seja, depende da arquitetura projetada para o *software*.

2.3 Projetos de *Software* Reutilizável

Três grandes classes de projeto de *software* têm como objetivo o reuso de código para a construção de novos projetos: ambientes de programação, bibliotecas de software e arcabouços de software.

2.3.1 Bibliotecas de *Software* (*Toolkit*)

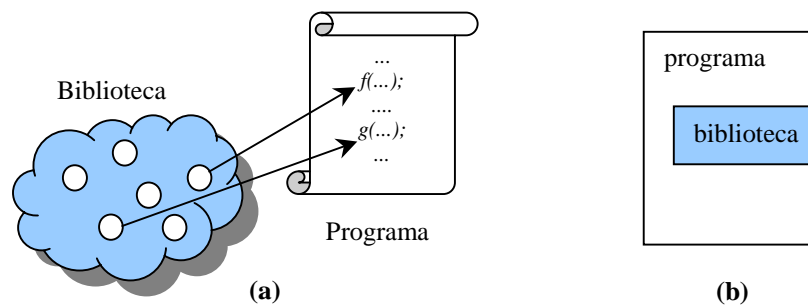


Figura 1. Relação entre uma biblioteca e o programa que a reutiliza: (a) tempo de compilação – a biblioteca não define a arquitetura da aplicação resultante, o programa escrito pelo usuário chama rotinas da biblioteca; e (b) tempo de execução – as rotinas escritas pelo usuário são a principal parte do processo.

Uma biblioteca é um conjunto de sub-rotinas, classes e objetos inter-relacionados projetado para fornecer uma determinada funcionalidade. As bibliotecas não impõem uma arquitetura de *software* específica às aplicações que a utilizam. Quando o programador faz uso de uma

biblioteca, ele escreve o código principal da aplicação e chama o código que deseja reutilizar (Gamma et al. 1994, p. 41). Exemplos de bibliotecas são a *Standard Template Library* distribuída em qualquer ambiente ANSI C++ e a biblioteca para geoprocessamento *TerraLib* (Câmara e Souza et al. 2000) desenvolvida pela Divisão de Processamento de Imagens do INPE. A figura 1 mostra a relação entre uma biblioteca e o programa que a reutiliza.

2.3.2 Arcabouços de *Software* (*Framework*)

Um arcabouço de *software* é um conjunto de sub-rotinas, classes e objetos cooperantes que constituem um projeto reutilizável para um domínio de aplicação específico que captura as decisões de projeto que são comuns a esse domínio (Schmidt et al. 1996; Kobryn 2000; Gamma et al. 1994, p.42; Buschmann et al. 1996, p. 396). A figura 2 ilustra a relação entre um arcabouço de *software* e o programa que o reutiliza.

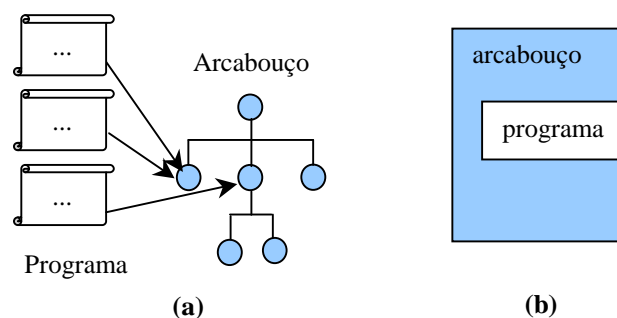


Figura 2. Relação entre um arcabouço e o programa que o reutiliza: (a) tempo de compilação – o arcabouço possui uma arquitetura definida que determina a arquitetura da aplicação resultante, o arcabouço chama rotinas escritas pelo usuário; e (b) tempo de execução – as rotinas escritas pelo usuário são apenas uma pequena parte do processo.

Um arcabouço de *software* dita a arquitetura da aplicação que o reutiliza. Ele irá definir a estrutura geral da aplicação, sua divisão em classes e objetos e, conseqüentemente, as principais responsabilidades das suas classes, como estas colaboram e o fluxo de controle. Esses parâmetros de projeto são predefinidos, de maneira que o programador ou projetista

da aplicação possa se concentrar nos aspectos específicos da sua aplicação (Gamma et al. 1994, p.42).

O uso de um arcabouço de *software* leva a uma inversão de controle entre aplicação e o software reutilizado. De forma diferente do que acontece no uso de bibliotecas, quando um programador usa um arcabouço, ele reutiliza o corpo principal da aplicação e escreve o código que este chama (Gamma et al, 1994, p. 42).

2.3.3 Ambientes de Programação

Ambientes de programação consistem de uma linguagem de programação de alto nível, um compilador que traduz códigos nessa linguagem para uma linguagem intermediária mais simples e uma máquina que, possui algoritmos e estruturas de dados que implementam as características da linguagem de alto nível e é capaz de executar códigos escritos na linguagem intermediária. A figura 3 ilustra a relação entre o programa do usuário e o ambiente de programação.

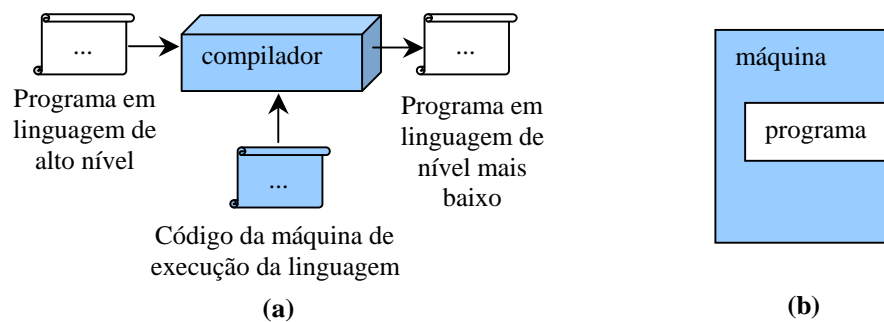


Figura 2. Relação entre um ambiente de programação e o programa que o reutiliza: (a) tempo de compilação – o usuário escreve o programa utilizando uma linguagem de alto nível e o compila para gerar um programa equivalente em uma linguagem de nível mais baixo; e (b) tempo de execução – a máquina de execução do ambiente executa o programa do usuário.

Quando um programador escreve um programa em uma determinada linguagem ele está, na verdade, reutilizando o código presente na máquina que a executa. Quando um

interpretador é utilizado no lugar do compilador, ele precisa implementar as funcionalidades tanto da máquina de execução da linguagem como do compilador.

2.3.4 Discussão

A interface de um ambiente de programação com seu cliente é definida na forma de uma linguagem de programação de alto nível. Portanto, a expressividade e a legibilidade do código são máximas quando comparados com os códigos das bibliotecas e dos arcabouços de *software*. Todavia, mesmo pequenas alterações léxicas, sintáticas ou semânticas na linguagem de programação implicariam em modificações complexas em seus analisadores léxicos, sintáticos e semânticos. Por esta razão, ambientes de programação são considerados pouco flexíveis. A linguagem de programação deve prover mecanismos que permitam a extensão do seu sistema de tipo para que o ambiente de programação possa ser considerado extensível.

A construção de um ambiente de programação é uma tarefa difícil e dispendiosa. A construção de arcabouços de *software* é simplificada por não envolver a construção de um compilador. A construção de uma biblioteca é ainda mais fácil. Porém, arcabouços de *software* costumam ser mais expressivos e legíveis que bibliotecas porque capturam decisões de projeto relevantes ao domínio para o qual foram desenvolvidos.

2.4 Reuso de Código na Programação Orientada a Objetos

Antes de dar início ao estudo das facilidades oferecidas pelas linguagens de programação orientadas por objetos para o desenvolvimento de códigos reutilizáveis e para o reuso de código, é preciso fazer uma clara distinção entre as noções de tipo e de classe. Da consciência das diferenças existentes entre estes dois conceitos surgirá um projeto de *software* de maior reusabilidade.

2.4.1 Os conceitos de tipo e classe

A classe de um objeto define como ele é implementado, ou seja, define como o estado interno do objeto é representado, que algoritmos são utilizados no desenvolvimento de seus serviços e como são implementados. O tipo de um objeto refere somente a sua **interface**, isto é, o conjunto das assinaturas dos métodos que implementam todos os serviços oferecidos pelo objeto. A assinatura de um método especifica o número de parâmetros que ele recebe como entrada, o tipo de cada parâmetro e tipo do valor de retorno. O tipo de um objeto é o nome utilizado para dar nome a uma interface específica (Gamma et al. 1994, p. 29).

Linguagens como C++, ao contrário de Java, utilizam classes tanto para a especificação do tipo do objeto como da sua implementação. Em C++, o programador implementa uma interface através da definição de uma classe abstrata. Uma classe abstrata é uma classe que posterga toda ou parte da implementação para suas subclasses e que, portanto, não pode ser instanciada. Os métodos que uma classe abstrata declara, mas não implementa, são chamados métodos abstratos.

Naturalmente, existe uma forte relação entre as noções de classe e tipo. Dizer que um objeto é uma instancia de uma classe é o mesmo que dizer que o objeto suporta a interface definida pela classe (Gamma et al. 1994, p. 32).

2.4.2 Reuso de código por meio de herança e composição de objetos

As duas técnicas mais comuns para a reutilização de código em sistemas orientados a objetos são **herança** e **composição de objetos**. O mecanismo de **herança** permite que a implementação de uma classe seja definida em termos da implementação de outra. A herança é implementada pelo ambiente de execução de uma linguagem orientada por objeto e é facilmente definida pelo programador em tempo de compilação. Na **composição de objetos**, um serviço mais complexo é construído através do uso de serviços mais simples de vários objetos. A **composição de objetos** é implementada pelo programador e é definida,

de forma dinâmica e em tempo de execução, pela obtenção de referências para outros objetos por um determinado objeto.

A figura 4 apresenta a simbologia utilizada para representar herança e composição de objetos na notação UML – *Unified Modeling Language* (UML Revision Task Force 1999).

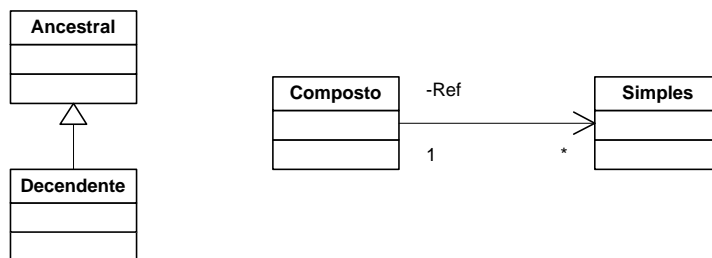


Figura 4. Notação UML para: (a) herança – uma linha vertical que parte da subclasse e tem triângulo na extremidade ligada à superclasse; (b) composição de objetos – uma seta com uma linha cheia indica que sua classe origem mantém uma referência para a classe destino. A referência tem um nome opcional, neste caso, “Ref”.

Essas duas formas de reuso possuem vantagens e desvantagens. O uso do mecanismo de herança é mais cômodo para o programador. Contudo, na herança a classe descendente geralmente tem visibilidade de toda a implementação da classe ancestral. Esse fato propicia que pelo menos parte da representação física da subclasse seja definida na classe ancestral. Desta forma, o acoplamento entre as classes se torna tão forte que alterações no código da classe ancestral podem exigir modificações de código da subclasse. Na herança, não é possível alterar as implementações utilizadas pela classe ancestral em tempo de execução, pois a herança é definida em tempo de compilação.

Na composição de objetos, a classe que constrói seu serviço pela composição de serviços mais simples, não tem visibilidade sobre as implementações dos mesmos e desta maneira, qualquer objeto pode ser substituído por outro objeto em tempo de execução, contanto que tenham do mesmo tipo. Porém, a composição de objetos deve ser implementada pelo programador.

Delegação é uma maneira de tornar a composição tão poderosa para fins de reutilização quanto a herança. Na delegação, dois objetos estão envolvidos no tratamento de uma requisição de serviço: um objeto *receptor* que repassa a requisição para um objeto *delegado* que a executa. Para que o objeto delegado possa se referir ao objeto receptor, da mesma maneira que as subclasses referenciam a superclasse, o objeto receptor passa a si próprio como um parâmetro da operação delegada. Entretanto, a delegação torna o *software* que a utiliza mais difícil de ser entendido e menos eficiente (Gamma et al. 1994, p. 35-36).

Neste momento, é preciso fazer uma distinção entre herança de classe e herança de interface. A **herança de classe** define a implementação de um objeto em termos da implementação de outro objeto. Resumidamente, é um mecanismo de compartilhamento de código e de representação. A **herança de interface** descreve quando um objeto pode ser utilizado no lugar do outro, isto é, define o tipo do objeto descendente como um subtipo do tipo do objeto ancestral. Em C++, herança implica tanto em herança de classe como herança de interface. Uma maneira de se utilizar somente herança de interface em C++ é definir somente heranças a partir de classe cujos métodos são puramente virtuais, isto é, classes abstratas puras.

A utilização exclusiva de herança de interfaces resulta em grandes benefícios para a reusabilidade de um projeto de *software*, pois os clientes que reutilizam seu código permanecem sem o conhecimento sobre as classes que implementam as funcionalidades utilizadas e este fato, reduz drasticamente as dependências de implementação entre subsistemas do projeto. Deste modo, sempre que possível, para privilegiar a reusabilidade do código, deve-se utilizar heranças de interface ao invés heranças de classe.

Portanto, um projeto de software reutilizável com metodologia orientada por objetos deve estabelecer uma combinação judiciosa entre herança e delegação, mantendo dois princípios básicos: (a) herdar preferencialmente de classes abstratas; (b) usar a delegação com parcimônia para evitar o crescimento exagerado do número de classes.

2.5 O uso de padrões para aumentar a reusabilidade de um projeto de *software*

Um especialista quando trabalha em um problema em particular, geralmente, procura algum problema semelhante entre os problemas para os quais ele conhece a solução. Se ele encontra, ele reutiliza a solução do problema encontrado. Somente se não o encontra, o especialista tenta inventar uma solução. Com base nessa idéia, os padrões de *software* foram criados.

Um **padrão** de *software* descreve um problema comum e recorrente encontrado no projeto arquitetural de um *software* que, surge em um contexto específico e que possui um esquema genérico e bem testado para sua solução. O esquema da solução é especificado pelos seus componentes, cada um com suas responsabilidades e relacionamentos, e pela maneira na qual eles colaboram (Buschman et al. 1996, p.8).

De acordo com Gama et al., (1994), um padrão nomeia, abstrai e identifica os aspectos-chaves de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado por objetos reutilizável. O padrão de *software* identifica as classes e objetos participantes da estrutura, seus papéis, colaborações e a distribuição de responsabilidades. Cada padrão descreve quando deve ser aplicado, se ele pode ser aplicado em função de outras restrições de projeto e as conseqüências, custos e benefícios de sua utilização.

Um padrão documenta uma experiência de projeto de *software* bem conhecida. Eles não são criados artificialmente. Ao invés disso, eles provêm um meio de reutilizar um conhecimento de projeto ganho pela experiência prática (Gamma et al. 1993) e, desta maneira, promovem o reuso de código.

Padrões provêm um vocabulário e entendimento comum para os princípios do projeto de *software* (Gamma et al. 1993). Logo, padrões melhoram a documentação de um *software* e facilitam a discussão sobre decisões de projeto. A melhoria da documentação é

especialmente importante para um arcabouço de *software*. Arcabouços têm uma curva de aprendizado acentuada que precisa ser percorrida antes deles se tornarem úteis (Gamma et al 1994, p. 42) citando (Beck e Johnson 1994).

Ademais, um padrão também é uma forma de documentar arquiteturas de *software* (Buschman et al. 1996, p.6). Portanto, facilitam a evolução e manutenção do software.

Buschman et al. (1996, p. 12-14) classificam os padrões de acordo com a escala das decisões arquiteturais a que si aplicam:

- **Padrões arquiteturais** lidam com a organização estrutural e fundamental de um sistema de computação e preocupam-se com os relacionamentos entre os objetos de maior granularidade no sistema. Eles dividem o sistema em subsistemas ou componentes de *software* predefinidos, especificam suas responsabilidades e incluem regras para organizar os relacionamentos.
- **Padrões de projeto** fornecem esquemas para refinar os subsistemas ou componentes de um sistema de computação, assim como, refinar os relacionamentos entre eles. Eles descrevem uma estrutura de comunicação comum e recorrente que resolve um problema de projeto geral em um contexto particular.
- **Idiomas** são padrões, de baixo nível de abstração, específicos para uma determinada linguagem de programação. Eles descrevem como determinados aspectos de um componente ou de um relacionamento devem ser implementados em uma dada linguagem.

Não há consenso se os padrões devem ser aplicados a um projeto de *software* durante a fase de análise de requisitos, de projeto ou de implementação. Entretanto, existe uma regra que diz que os padrões arquiteturais devem ser aplicados mais cedo que os padrões de projeto que, devem ser aplicados antes dos idiomas (Buschman et al. 1996, p. 394).

O padrão arquitetural *Layer* (Buschman et al. 1996, p. 35-51) organiza um sistema em níveis de abstrações diferente, onde níveis mais altos são construídos sobre os serviços oferecidos pelo nível imediatamente inferior, sem que o nível mais alto saiba como esses serviços foram implementados. O idioma *Handle-Body* ou *Bridge* (Gamma et al. 1994, p. 151-159) em conjunto com o idioma *Counted Body* (Buschman et al. 1996, p. 345-358) podem ser utilizados para desacoplar as interfaces das implementações dos objetos em projetos que utilizem a linguagem C++, de forma que essas interfaces sirvam como referências para os objetos que designam e, de maneira a evitar que cópias indesejadas desses objetos sejam criadas ou deixadas na memória. Conseqüentemente, esses padrões aumentam a flexibilidade de um determinado código.

O padrão arquitetural Reflexão (Buschman et al. 1996, p. 193-219) provê um mecanismo para mudar a estrutura e o comportamento de um sistema dinamicamente. Ele suporta a modificação de aspectos fundamentais como estrutura de tipo e mecanismos de chamada de função. Neste padrão, uma aplicação é dividida em duas partes. Uma parte fornece meta informações sobre propriedades do sistema e faz com que o sistema tenha conhecimento da sua própria estrutura. A outra parte implementa a lógica da aplicação sobre a primeira parte. Mudanças na parte das meta informações afetam diretamente a outra parte. Este padrão arquitetural pode é utilizado na construção de sistemas de computação extensíveis.

2.6 Contribuições da programação genérica para a reusabilidade de um projeto de software

Programação genérica (Austern, 1998) se refere a uma forma de abstração usada no processo de desenvolvimento de *software* na qual se procura identificar as propriedades funcionais e estruturais fundamentais de um algoritmo ou de um tipo abstrato de dado, de forma que eles possam ser construídos sem que haja dependência em relação aos tipos sobre os quais serão definidos, e desta maneira, mais reutilizáveis. Por exemplo, um algoritmo para ordenar inteiros funciona de forma equivalente a um algoritmo para ordenar

caracteres e uma árvore binária de números inteiros possui estrutura e funcionalidades equivalentes a uma árvore binária de caracteres.

O objetivo do paradigma de programação genérico é a implementação de algoritmos e tipos abstratos de dados que recebam como parâmetro os tipos sobre os quais são definidos e, além disso, garantir a eficiência dessas implementações.

A contribuição da programação genérica para o desenvolvimento de software reutilizável tem duas grandes vertentes:

- A disponibilidade de biblioteca de algoritmos e estruturas de dados genéricos, como a STL (Standard Template Library) da linguagem C++. Estas bibliotecas reduzem muito o esforço de programação, ao fornecer algoritmos com qualidade para problemas típicos de programação, como ordenação e busca em listas.
- A combinação da programação genérica com programação orientada por objetos, no contexto de programação multi-paradigma, ou seja, a combinação de diferentes estilos de programação (Coplien 1998)

A idéia de se combinar classes parametrizadas, mecanismos de herança e padrões para melhoria da reusabilidade de *software* é explorada em (Coplien 1996). No esquema por ele proposto, um padrão é ser expresso como uma classe genérica e seu reuso se realizaria através do mecanismo de herança, em dois passos: (1) uma classe base é definida como um padrão genérico; (2) a classe desejada é definida como uma especialização da classe base. Veja figura 5(a).

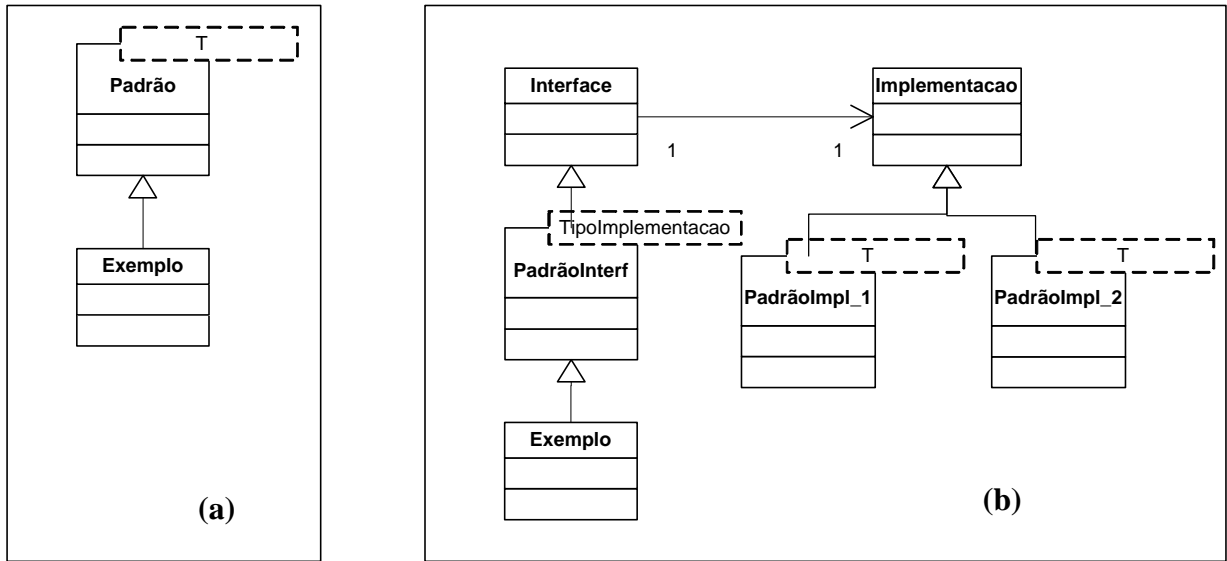


Figura 5. Exemplo de uso conjunto de programação genérica e padrões para melhoria da reusabilidade de um código: (a) esquema geral sugerido por Coplien (1996); e (b) esquema de maior reusabilidade.

Um algoritmo de ordenação genérico possui a mesma implementação quando definido sobre os tipos inteiro e caractere. Da mesma forma, uma árvore binária genérica possui a mesma implementação quando definida sobre esses dois tipos. Portanto, podemos afirmar que a programação genérica pura não desacopla a abstração de um tipo abstrato de dados ou de um algoritmo da sua implementação.

Conseqüentemente, no esquema da figura 5(a), uma implementação de um padrão não poderia ser substituída por outra implementação do mesmo padrão em tempo de execução, nem poderia ser utilizada no lugar da outra em tempo de compilação, apesar de possuírem a mesma *interface*. Isto é verdade porque, neste esquema, as duas implementações possuem a mesma interface, mas tipos diferentes.

Portanto, o esquema proposto poderia gerar padrões genéricos de maior reusabilidade, se pudesse ser usado para gerar padrões onde as abstrações fossem separadas das implementações. Para isso, é preciso acreditar que toda implementação de um padrão possui uma interface e então, separá-la da implementação, como mostra a figura 5(b). Esta

interface poderia ser entendida como uma representação possível de uma implementação daquele padrão.

No esquema da figura 5(b), a herança de classe presente no esquema da figura 5(a) é substituída por uma herança de interface, como sugere a seção 2.4. A abstração do próprio padrão é separada da implementação e a única interface de padrão passa a contar com várias implementações, todas do mesmo tipo: *Implementação*.

CAPÍTULO 3

Modelos Dinâmicos Espaciais

3.1 Introdução

Segundo (Smyth 1988, p.192 citado em Couclelis 2000), um modelo para um fenômeno espacial dinâmico pode ser pensado como um micro-mundo definido por uma ontologia que consiste em um conjunto de entidades que o habitam, uma estrutura temporal, uma estrutura espacial, regras de comportamento e uma lógica. As entidades caracterizam a paisagem do micro-mundo, alguns exemplos são: corpos d'água, diferentes categorias de uso do solo, estradas, etc. A escolha das entidades que farão parte de um modelo depende, principalmente, do intuito com o qual o modelo será construído e do domínio ao qual ele se aplicará. A estrutura espacial e temporal de modelos geográficos é assunto de várias pesquisas (Engenhofer e Colledge 1998; Parent 1999; Câmara et al. 2000; Tryfona 2000; Bennett 2001; Peuquet 2001), diferentes tipos de conceitualizações e arcabouços foram propostos para modelar o tempo e o espaço e algumas serão discutidas ao longo deste capítulo. As regras de comportamento definem como as diversas entidades do micro-mundo poderiam evoluir e interagir, isto é, elas definem os possíveis comportamentos de um modelo. A lógica de um modelo, também chamada regras de inferência, define que fatos podem ser deduzidos a partir de uma dada configuração do micro-mundo e como estas fatos podem ser deduzidos. Nos modelos matemáticos, a lógica do modelo é indistinguível das regras de comportamento, pois ambas estão implícitas no formalismo usado.

De acordo com esta concepção, a construção de um modelo espacial dinâmico pode ser separada em quatro partes: (a) a definição de uma representação computacional para o espaço; (b) a escolha das entidades que farão parte do modelo e a conversão de dados sobre essas entidades para um formato adequado ao modelo espacial definido no item anterior; (c) a escolha de uma representação para o tempo; e (d) a construção de modelos que

simulem o comportamento de sistemas reais que alteram os atributos das entidades em localizações específicas conforme o tempo evolui. Por esta razão, o restante desse capítulo discute as representações computacionais para espaço, tempo e comportamento e procura identificar uma estrutura comum aos modelos espaciais dinâmicos.

3.1.1 O paradigma dos quatro universos

Para abordar as representações computacionais das componentes espaço, tempo e comportamento, as demais seções deste capítulo utilizam o mesmo arcabouço conceitual utilizado em (Câmara et al. 2000) e desenvolvido em (Gomes e Velho, 1995): o “paradigma dos quatro universos”, figura 6. Este mecanismo geral de modelagem faz distinção entre quatro diferentes níveis de abstração:

- o universo do *mundo real*, que inclui as componentes da realidade a serem modeladas;
- o universo *matemático (conceitual)*, que inclui uma definição matemática (formal) das componentes a ser representadas;
- o universo de *representação*, onde as diversas componentes contínuas formais são discretizadas e mapeadas para representações geométricas e alfanuméricas no computador;
- o universo de *implementação*, onde as estruturas de dados e algoritmos são escolhidos, baseados em considerações como desempenho, capacidade do equipamento e tamanho da massa de dados. É neste nível que acontece a codificação.

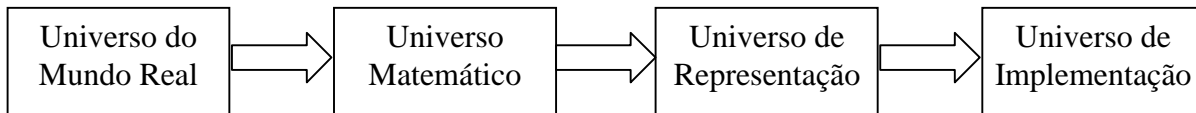


Figura 6. O paradigma dos quatro universos.

3.2 Espaço

Para os geógrafos existe uma distinção entre os conceitos de espaço absoluto e espaço relativo. “Espaço absoluto, também chamado Cartesiano ou Newtoniano, é um container de coisas e eventos, uma estrutura para localizar pontos, trajetórias e objetos. Espaço relativo, ou Leibnitziano, é o espaço constituído pelas relações espaciais entre coisas e eventos” (Couclelis 1997). (Santos 1996) refere-se a distinção entre espaço absoluto e espaço relativo como o “espaço dos fixos” e o “espaço dos fluxos”.

Couclelis (1997) propõe a idéia de espaço próximo como uma extensão dos conceitos de espaço absoluto e relativo. No espaço próximo, a informação espacial é geo-referenciada como no espaço absoluto, mas a cada localização é anexada uma representação do espaço relativo da qual ela faz parte. Enquanto no espaço absoluto o principal conceito é o item geo-referenciado e no espaço relativo é a relação espacial, o principal conceito no espaço próximo é a vizinhança.

No que diz respeito ao universo matemático, Câmara et al. (2000) define um arcabouço unificado para os problemas de modelagem de entidades espaciais, no qual uma definição matemática geral para objetos espaciais é proposta e, em seguida, mostra que diferentes tipos de dados espaciais podem expressos como casos particulares desta definição. Especificamente, um objeto espacial foi definido como uma tripla $so = (S, A, f)$ onde:

1. $S \subseteq \hat{A}^2$ é um subconjunto do plano Euclidiano, que é o suporte geométrico do objeto espacial.

2. A é um conjunto de domínios de atributos A_1, \dots, A_n .

3. $f: S \rightarrow A_1 \times A_2 \times \dots \times A_n$ é a função de atributo do objeto espacial, que associa a cada local do suporte geométrico um valor do conjunto de domínios de atributos.

Um objeto espacial é geo-referenciado se existe uma parametrização g do suporte geométrico S para a superfície da terra. Matematicamente, o mapeamento g pode ser descrito aproximadamente por uma parametrização de S para superfície de uma esfera.

Segundo Câmara et al. (2000), a representação de um objeto espacial é realizada pela discretização tanto do seu suporte geométrico quanto da sua função de atributo e duas possíveis representações do espaço são destacadas: matricial e vetorial. Na representação matricial o espaço é representado como uma matriz de células composta de m colunas e n linhas, onde cada célula possui coordenadas que a endereçam e um valor correspondente ao seu único atributo. Na representação vetorial, a representação de um objeto é uma tentativa de reproduzi-lo o mais exatamente possível. A representação gráfica de qualquer objeto vetorial é reduzida a três formas básicas: pontos, linhas ou polígonos. A representação matricial pode ser especializada em grades regulares, modelos numéricos de terreno e imagens. A representação vetorial pode ser especializada em estruturas do tipo arco-nó e arco-nó-polígono.

Em Câmara et al. (2000), o suporte geométrico e a função de atributo de um objeto espacial também são utilizados para classificar os objetos espaciais em diferentes categorias de dados geográficos que englobam os conceitos tradicionais de campos, objetos e planos de informação encontrados nos SIGs: objetos espaciais simples, objetos espaciais compostos e homogêneos, objetos simples e não homogêneos e objetos espaciais compostos e não homogêneos. Representações matriciais são adequadas à representação de dados geográficos simples e não homogêneos, as demais categorias são melhor representadas por estruturas vetoriais.

Um espaço celular é um tipo de objeto espacial que representa o conceito de espaço próximo e que é de especial interesse para a modelagem dinâmica. Este objeto tem como

suporte geométrico uma grade de células regulares ou não e uma função de atributo que a cada célula associa vários atributos. Entre as categorias de dados geográficos propostas em (Câmara et al. 2000), ele é melhor classificado como um objeto composto e não homogêneo, onde cada célula possui um atributo especial para representar sua vizinhança. Desta maneira, uma representação matricial não é adequada para um espaço celular, sua representação mais direta se dá na forma vetorial, onde cada célula pode ser um polígono, uma linha ou um ponto.

3.3 Tempo

Apesar de ser reconhecida a necessidade de incorporar a dimensão temporal em muitos processos ambientais, a representação do tempo em SIGs não passou de um estágio embrionário (Engenhofer et al. 1999). Isto se deve (1) ao paradigma cartográfico sobre o qual os SIGs foram construídos, (2) a ênfase dada em soluções orientadas a implementação, e (3) a ausência de uma teoria espaço-temporal (Peuquet 2001).

Conceitualmente, pode-se representar o tempo através de diferentes estruturas, definidas, principalmente, com base em três aspectos da representação temporal: granularidade, variação e ordem no tempo (Figura 7).

A ordem temporal refere-se ao modo como o tempo flui. Neste caso, pode-se assumir que o tempo flui de forma linear, ramificada ou cíclica. No tempo linear considera-se que o tempo flui seqüencialmente, ou seja, existe uma ordem de precedência entre os pontos no tempo, de tal forma que cada ponto tenha apenas um sucessor e um antecessor. No tempo ramificado, múltiplos pontos podem ser os sucessores ou antecessores imediatos de um mesmo ponto. O tempo cíclico é utilizado para modelar eventos e processos recorrentes. Associado ao conceito de variação temporal discreta, existe o conceito de *Chronon*. Um *chronon* é a menor duração de tempo suportada por um modelo e pode variar em diferentes aplicações (Edelweiss e Oliveira 1994).

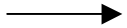


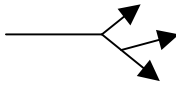


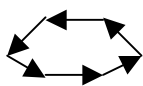

Ordem no tempo	Variação Temporal	Granularidade
Linear 	Discreto 	Instante 
Ramificado 	Contínuo 	Intervalo 
Cíclico 		Período 

Figura 7 - Estruturas temporais (fonte: Worboys, 1998)

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

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

A representação do tempo em modelos espaciais dinâmicos não se limita a uma simples questão de estender os SIGs para incorporar conceitos de banco de dados temporais. Na dimensão temporal assim como na dimensão espacial do modelo, a dicotomia contínuo-discreto é uma questão desafiante. Eventos tais como relâmpagos e erupções vulcânicas são

discretos tanto no domínio temporal quanto no espacial, enquanto temperatura e precipitação são processos espaço-temporais contínuos (Peuquet 2001). Outro conceito forte em sistemas temporais refere-se à dinâmica de atualização dos objetos, que pode ser síncrona ou assíncrona. Em sistemas síncronos todos os elementos do sistema são atualizados simultaneamente (Sipper 1999). Já em sistemas assíncronos, os elementos não são atualizados em intervalos de tempo regulares.

Para representar o tempo, a máquina que coloca um modelo em funcionamento deve manter uma variável global que a cada passo da simulação é incrementada. Quando o tempo é considerado discreto, uma variável numérica é utilizada para representar o tempo e a cada passo da simulação é incrementada de um *chronon*. Nesta abordagem, o tempo pode fluir somente de forma linear e discreta. Porém, técnicas de simulação discreta dirigida por eventos (Jain 1991) permitem que o tempo possa ser tratado tanto como uma variável contínua como discreta e permite que o tempo flua de forma linear, ramificada ou cíclica.

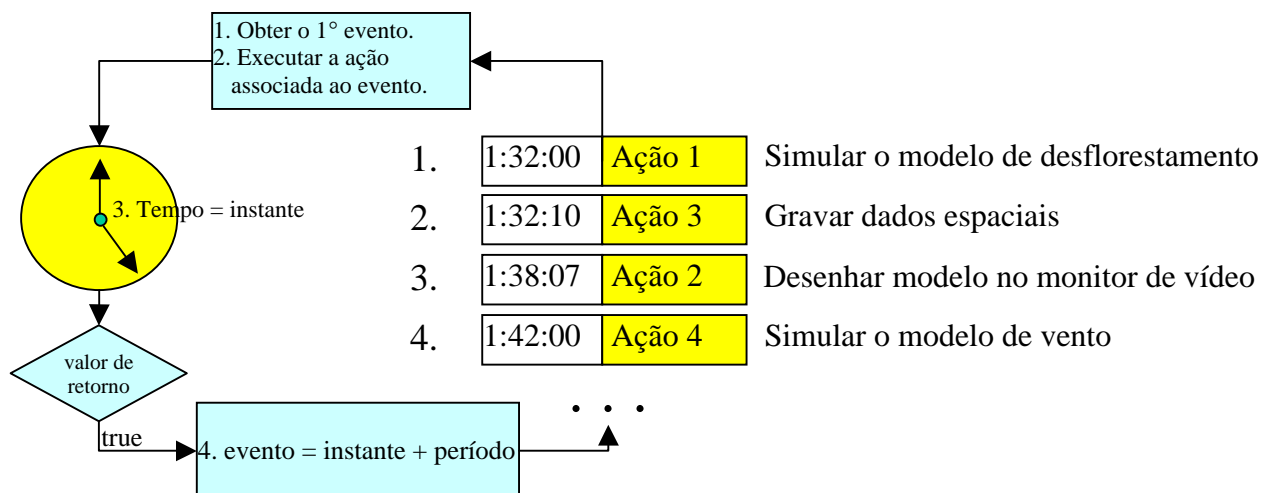


Figura 8. Esquema de um escalonador de tempo dirigido por eventos discretos.

Um simulador baseado em eventos discretos possui a estrutura apresentada na figura 8: (1) cada evento é definido pelo instante em que deve ocorrer e por um valor que indica período no qual ele ocorre; (2) um escalonador de eventos mantém os eventos ordenados pelo

instante em que devem ser disparados; (3) um mecanismo de avanço de tempo – sempre que o evento na cabeça da fila de eventos é disparado, a variável global tempo é atualizada automaticamente para o instante que o próximo evento na fila deverá ocorrer; e (4) ações associadas aos eventos – cada evento é simulado por uma rotina que atualiza as variáveis que definem o estado do sistema e, provavelmente, inserem outros eventos na fila do escalonador. Essas rotinas possuem um valor de lógico retorno, se uma rotina retorna verdadeiro o evento a ela associado é re-inserido no escalonador, caso contrário não. É importante ressaltar que o termo “discreto” não se aplica aos valores de tempo utilizados na simulação, um simulador discreto dirigido por eventos pode utilizar valores de tempo contínuos e discretos.

A grande maioria dos modelos espaciais dinâmicos encontrados na literatura (Veldkamp e Fresco 1996; Verburg et al. 2002; Soares-Filho 2002; White et al. 1997; Lim et al. 2002) consideram o tempo como uma variável discreta, somente o modelo (Pedrosa et al. 2002) considera tanto os fenômenos discretos quanto contínuos.

3.4 Comportamento

Modelos comportamentais podem ser classificados segundo a teoria formal a eles subjacente ou segundo a ausência de uma teoria (Briassoulis 2000; Veldkamp e Lambin 2001). Modelos de processos (*theory-driven models*) podem ser entendidos como aqueles cujas suposições, premissas e equações derivadas que definem o comportamento do sistema sendo modelado são estabelecidas “a priori”. Modelos empíricos (*data-driven models*) baseiam-se nos dados disponíveis para esboçar conclusões “a posteriori” sobre o sistema. Alguns modelos podem ser chamados de modelos híbridos, por utilizarem ambas as abordagens. O modelo proposto por White et al. (1998) define acessibilidade com base na teoria gravitacional e estima, através de equações probabilísticas, a adequação de uma determinada região do espaço a mudanças do seu uso original para outro tipo de uso.

Os modelos da primeira categoria possuem uma base teórica, que usualmente consiste num conjunto de equações matemáticas que devem ser parametrizadas para caracterizar o fenômeno estudado. Neste sentido, tratam-se de aproximações gerais, e os problemas estudados serão sempre caracterizados como casos particulares do modelo. Modelos teóricos são geralmente construídos para servirem como ferramentas explanatórias e, desta maneira, os resultados são freqüentemente generalizáveis para uma gama de aplicações (Parker et al, 2001). Modelos teóricos são capazes de lidar com a heterogeneidade temporal, isto é, mudanças fundamentais que podem ocorrer nas forças direcionadoras ou nos processos ao longo do tempo (Velkamp e Lambin 2001).

Os modelos empíricos não procuram explicar o fenômeno sob estudo ou sua causa e se baseiam na capacidade de deduzir implicitamente as leis que regem os fenômenos a partir da coleta de dados descritivos. Em função do processo de coleta e da necessidade de predição multi-temporal a partir de dados históricos, estes modelos usualmente supõem que os processos de mudança são estacionários. Modelos empíricos são geralmente projetados para se ajustarem a detalhes de um caso de estudo particular e, desta maneira, suas conclusões são usualmente específicas para esse caso.

Enquanto no universo real coexistem os comportamentos dos sistemas ecológicos, sócio-econômicos e físicos, a construção de modelos teóricos utiliza conceitos que se baseiam fortemente na teoria de sistemas (Lee e Varayia 2003) que enfatiza a modelagem de sistemas complexos através de princípios matemáticos que suportam o projeto baseado em hierarquias e a decomposição dos sistemas em componentes. Para sistemas discretos, os métodos matemáticos incluem lógica e máquinas de estados. Para sistemas contínuos, os métodos matemáticos incluem álgebra linear e equações diferenciais.

De uma maneira geral, os modelos comportamentais possuem três componentes chaves, como mostra a figura 9: uma configuração espacial inicial do espaço, uma função de transição, também chamada função de mudança, e uma configuração espacial de saída. A

configuração inicial de um modelo dinâmico pode ser obtida a partir de dados históricos do fenômeno em estudo.

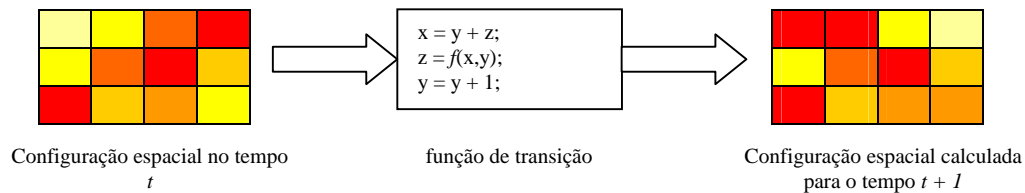


Figura 9. Visão geral de um modelo comportamental.

Em (Lambim 1994) são discutidos três modelos empíricos onde as funções de transição são implementadas de maneira diferente: cadeias de Markov, modelos logísticos de difusão e modelos de regressão. Em (Soares Filho 1998), modelos baseados em cadeia de Markov e em modelos logísticos de difusão são empregadas na modelagem dinâmica da paisagem de uma região amazônica. Em Reis e Margulis (1991) um exemplo de modelo de regressão é usado para implementar a função de transição e modelar o desmatamento da Amazônia em função da densidade espacial das atividades econômicas da região.

A maneira mais fácil de se construir a função de transição para modelos teóricos é na forma de uma tabela de regras de transição, a partir de agora chamada tabela de transição. Para cada configuração possível dos atributos que caracterizam o espaço, existe uma entrada na tabela de transição que descreve as regras que devem ser aplicadas para aquele caso. As regras de transição podem ser construídas utilizando qualquer tipo de abordagem matemático-computacional: podem utilizar lógica booleana, lógica de predicados, lógica de segunda ordem ou lógica fuzzy; podem ser construídas com base em leis da física ou com base em um conhecimento prático sobre o fenômeno estudado; podem ser expressas como um sistema de equações diferenciais ou por como equações estocásticas; podem ser implementadas por algum método numérico ou de alguma heurística. A abordagem a ser escolhida depende do domínio do problema para o qual o modelo está sendo construído e da experiência do modelador.

As máquinas de estados têm sido largamente usadas na modelagem dinâmica devido a sua simplicidade e a existência de uma teoria que lhe dá suporte (Minsky 1967). Uma máquina de estados finitos ou autômato finito (Hopcroft e Ullman 1979) pode ser definida por um grafo dirigido $G_g = (V, E_g)$, chamado diagrama de transição, onde V é um conjunto finito de vértices e E_g é um conjunto de pares ordenados de vértices chamados arcos. Cada vértice do grafo corresponde a um estado do autômato. Se existe uma transição do estado q para o estado p do autômato em resposta a uma entrada a , então existe um arco com o rótulo a que parte do vértice correspondente ao estado q para o vértice correspondente ao estado p no diagrama de transição G_g . A cada arco do autômato existe uma regra de transição associada que é aplicada sobre o espaço. Portanto, a tabela de transição pode ser considerada como um caso particular de um autômato finito que possui um único estado interno, contendo somente arcos reflexivos.

A figura 10 mostra o diagrama de estados de uma máquina de estados finitos, chamada *memória*, capaz de armazenar um dígito binário que tenha sido fornecido como entrada no momento $t-1$. O símbolo de entrada que provoca uma transição é apresentado na origem do arco que representa essa transição. Os símbolos inseridos no meio dos arcos representam a resposta da máquina no momento de uma transição.

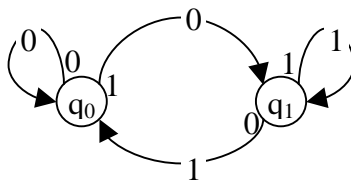


Figura 10. Diagrama de transição da máquina memória.

Um autômato finito pressupõe que o tempo evolui de forma discreta (Minsky 1967, p. 12). O comportamento dessas máquinas é descrito como uma seqüência simples e linear de eventos no tempo. Desta maneira, a variável t que representa o tempo assume os valores discretos $0, \pm 1, \pm 2, \dots$. Portanto, os autômatos finitos não são capazes de modelar comportamento contínuos.

Um autômato híbrido (Henzinger 1996) é um modelo para um sistema cujo comportamento possui componentes discretas e contínuas, isto é, um sistema híbrido. Um sistema híbrido pode ser visto como um sistema que possui um programa discreto que possui um ambiente contínuo (Alur et al. 1995). Um autômato híbrido consiste de um autômato finito equipado com variáveis que evoluem continuamente no tempo. Os estados discretos do sistema são modelados por um vértice no grafo dirigido do autômato finito, cada vértice é chamado de *modo de controle*. A dinâmica discreta do sistema é modelada pelos arcos do grafo, chamados de *transições de controle*. O estado contínuo é modelado por pontos em \mathfrak{R}^n e a dinâmica contínua do sistema é modelada por *condições de fluxo* como, por exemplo, equações diferenciais. O comportamento do sistema depende do *modo de controle* do autômato finito: cada modo de controle determina uma *condição de fluxo* e cada *transição de controle* pode causar mudanças no estado do sistema, como determinado por uma *condição de salto*. Da mesma maneira, o comportamento do autômato finito depende do estado do sistema: cada modo de controle continuamente avalia uma condição invariante do sistema e se esta condição é violada, uma mudança contínua no sistema terá causado uma *transição de controle*. Deste modo, uma máquina de estados finitos pode ser pensada como um caso particular de um autômato híbrido para o a componente contínua é ignorada.

O autômato híbrido da figura 11 modela um sistema de variação climática. A variável x representa a temperatura. No *modo de controle esfriando*, o clima está esfriando e a temperatura cai segundo a *condição de fluxo* $x' = -0,1x$. No *modo de controle esquentando*, o clima está esquentando e a temperatura sobe de acordo com a condição de fluxo $x' = 5-0,1x$. Inicialmente, a temperatura é igual a 20 graus. De acordo com a *condição de salto* $x < 19$, o processo climático pode passar a esquentar tão logo a temperatura caia abaixo de 19 graus. De acordo com a *condição invariante* $x \geq 18$, a temperatura pode continuar caindo até que o valor mínimo de 18 graus seja atingido.

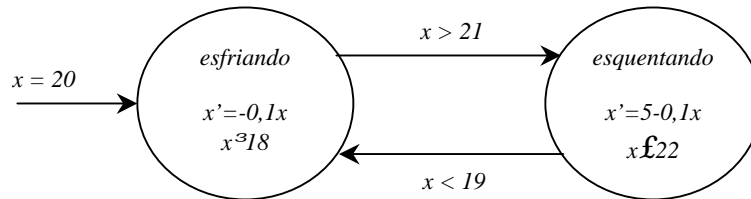


Figura 11. Um autômato híbrido para um sistema climático.

Algumas pesquisas supõem que a dinâmica observada dos sistemas complexos é um resultado em macro-escala de simples interações entre os componentes do sistema em micro-escala. Esta linha de pensamento levou ao surgimento dos modelos dinâmicos baseados em agentes. Um agente é qualquer coisa capaz de perceber seu ambiente através de sensores e agir sobre este ambiente através de atuadores. A noção de agente está relacionada à idéia de uma ferramenta para analisar sistemas, não é uma caracterização absoluta que divide o mundo em agentes e não-agentes (Russel e Norvig 1995). Agentes representam entidades animadas como, por exemplo, pessoas, animais, organizações sociais, empresas e governos locais, ou entidades inanimadas como, por exemplo, objetos físicos ou computacionais, que interagem na micro-escala e geram comportamentos complexos na macro-escala. Em (Russel e Norvig 1995), um agente reflexivo com estado interno é definido como algo que utiliza a sua percepção atual do ambiente e seu estado interno combinados para gerar um novo estado interno e decidir como agir sobre o ambiente. Desta maneira, este tipo de agente pode ser descrito como uma máquina de estados finitos para a qual o conjunto de todas as percepções possíveis do ambiente forma o alfabeto de entrada, as ações do agente são a saída da máquina e o estado interno do agente é um dos estados possíveis do autômato.

Um número crescente de pesquisadores vem explorando o potencial dos modelos baseados em agentes para a modelagem das decisões humanas. Por exemplo, estudos da dinâmica do uso e cobertura do solo consideram o comportamento humano um elemento crítico para essa dinâmica (Parker et al. 2001). Couclelis (Parker et al. 2001, p. 5-6) expressa seu ceticismo sobre a utilidade desses modelos com base em dois principais argumentos.

Primeiro, teorias formais para as interações humano-ambiente ainda não estão desenvolvidas e este déficit tem impedido a modelagem projetiva de fenômenos espaciais. Segundo, ela duvida que os benefícios explanatórios de um modelo combinado excedam o custo das, talvez exponencialmente ampliadas, complexidades das dinâmicas sociais e ambientais misturadas. Entretanto, Parker et al. (2001) contra-argumentam que modelos baseados em agentes podem servir como uma ferramenta para o desenvolvimento da teoria de processos sobre alguns fenômenos espaciais que ela diz ainda não constar na literatura. Enquanto uma teoria de processos ainda não foi desenvolvida, modelos sofisticados de processos individuais, tais como, a tomada de decisões humanas e o processo de difusão espacial, estão sendo desenvolvidos de maneira crescente. Um modelo de agentes oferece um meio de acoplar estes processos para desenvolver uma teoria integrada sobre relacionamentos causais. Outro contra-argumento apresentado em (Parker et al. 2001) é que o custo de desenvolvimento e análise de modelos tem caído radicalmente devido ao aumento do poder de computação, surgimento de ferramentas de *software* inovadoras e do desenvolvimento de plataformas especialmente projetadas para a modelagem baseada em agentes. Além disso, enquanto a obtenção de dados continua sendo um desafio, especialmente considerando dados socioeconômicos desagregados, no lado ambiental, a disponibilidade de imagens de satélite e o advento de técnicas de sensoriamento remoto têm relaxado as restrições sobre os dados disponíveis. Estratégias especiais de amostragem para modelos de mudança de uso e cobertura do solo (Berger et al. in press) assim como os novos desenvolvimentos em técnicas econométricas baseadas em máxima entropia (Howitt e Reynaud 2001) demonstram um potencial considerável para prover dados desagregados e consistentes.

3.5 Modelos Espaço-Temporais

Todos os modelos espaciais dinâmicos estudados, entre eles (Veldkamp e Fresco 1996; Verburg et al. 2002; Soares-Filho 2002; White et al. 1998; Lim et al. 2002), modelam o

espaço como um espaço celular e possuem uma estrutura funcional similar, como mostra a máquina de estados da figura 12.

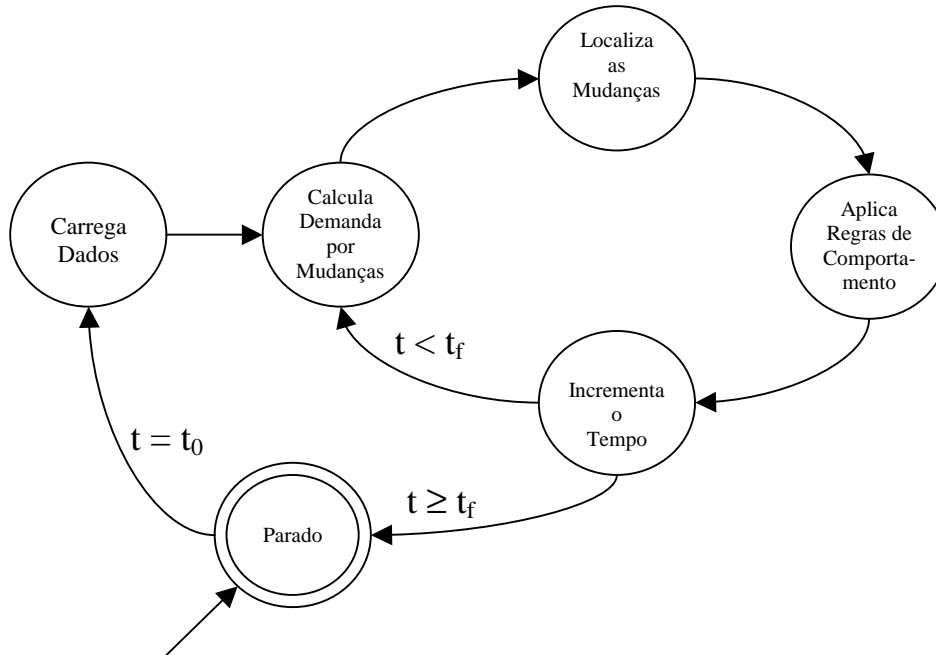


Figura 12. Estrutura comum aos modelos espaciais dinâmicos.

Nessa figura, o vértice denotado por duas circunferências concêntricas indica o estado inicial. No momento em que o modelador coloca o modelo em funcionamento, o instante inicial t_0 é atribuído à variável que registra o tempo t da simulação e então, os atributos das células referentes ao momento t_0 são carregados. Após a carga dos dados, tem início a fase de simulação propriamente dita. Alguns modelos possuem um primeiro estágio onde calculam a demanda por mudanças nos atributos das células, isto é, determinam que quantidade de células deve ter seus atributos alterados para satisfazer as exigências de uma determinada configuração do espaço. As duas abordagens mais comuns para esse cálculo são: (a) realizar uma comparação entre dados reais coletados, em instantes diferentes, sobre o fenômeno sob estudo e determinar a quantidade de mudança ocorrida (Veldkamp e Fresco 1996); ou (b) calcular a quantidade de mudança com base em algum outro modelo, por exemplo, um modelo demográfico ou econômico (White et al. 1997). Em um segundo estágio, os modelos precisam determinar onde as mudanças previstas deverão ocorrer, isto

é, é preciso escolher quais células terão seus atributos alterados para atender a demanda determinada no estágio anterior. A maneira mais comum de fazer esta escolha é calcular valores percentuais que indiquem para cada célula sua potencialidade para cada tipo de mudança e ordena-las segundo esses percentuais. O terceiro estágio da simulação consiste na aplicação das regras de comportamento do modelo sobre a estrutura celular para efetivamente realizar as mudanças. As regras de comportamento podem ser tão simples quanto alterações diretas nos atributos das primeiras células resultantes da ordenação realizada até que a demanda seja atendida, ou podem implicar na execução de heurísticas mais elaborada como os algoritmos *patcher* e *expander* definidos em (Soares-Filho 2002). Outros modelos podem ignorar os primeiros dois estágios da simulação e executar regras de comportamento baseadas em um modelo teórico (Wesseling 1996). Outros modelos podem adotar uma solução híbrida entre essas duas abordagens. No quarto estágio da simulação, a variável tempo é incrementada e se ela for menor que o momento final da simulação, a simulação retorna ao primeiro estágio e o ciclo se repete. Caso contrário, a simulação é terminada.

O modelo matemático para fenômenos espaço-temporais de maior destaque na literatura são os autômatos celulares (AC), que são assim considerados por serem tratáveis e apresentarem uma grande simplicidade operacional. Os ACs podem ser diretamente conectados a superfícies matriciais em um sistema de informação geográfica, são capazes de gerar dinâmicas que reproduzem processos de mudança através de difusão e contêm complexidade bastante para simular mudanças como aquelas observadas em fenômenos emergentes (Couclelis 1997; Wolfram 1984).

3.5.1 Autômatos Celulares

Um AC como concebido originalmente por Ulam e von Neumann nos anos 50 (von Neumann 1966, p. 106) consiste de um reticulado bidimensional e infinito de células quadradas, cada uma designada por um par de coordenadas inteiras i e j . Cada célula é ocupada por autômatos finitos equivalentes e cada autômato é conectado a seus quatro

vizinhos adjacentes. Portanto, as regras de transição são exatamente as mesmas para cada célula. Conseqüentemente, a estrutura celular é funcionalmente homogênea. Como cada autômato é conectado da mesma maneira a seus quatro vizinhos, a estrutura celular é também isotrópica¹. A homogeneidade funcional e a isotropia estão relacionadas com a estrutura, contudo, não existe relação com o conteúdo e estado das células. Por conseguinte, se células diferentes de uma região da estrutura celular estão em estados diferentes, uma parte desta região pode agir de uma determinada maneira e enviar informação em uma direção, enquanto outra parte da região pode agir de maneira diferente e enviar informação em outra direção. Um AC básico modela o tempo como uma variável síncrona e discreta, isto é, as regras de transição são avaliadas sobre todo o espaço de células em intervalos discretos e fixos de tempo (von Neumann 1966, p.100 e 133). Este modelo de dinâmica é conhecido como *snapshot time*.

Segundo (Wolfram, 1983), para ACs unidimensionais, uma célula é conectada a r vizinhos de cada lado, onde r é um parâmetro definido como raio. Desta maneira, cada célula possui $2r+1$ vizinhos incluindo a si mesma. Para um AC bidimensional dois tipos de vizinhança são considerados: (a) vizinhança de von Neumann - cinco células, que consiste da própria célula mais quatro outras não diagonais e imediatamente vizinhas; e (b) vizinhança de Moore - nove células, que consiste da própria célula mais oito outras imediatamente vizinhas. Contudo, essas definições de vizinhança foram alteradas para a noção de espaço próximo (Takeyana 1997). Em (O'Sullivan 1999) é proposta uma representação para o espaço próximo por meio de um *graphcellular automaton* (graph-CA), que estende um AC básico através do uso de um grafo direcionado G . A cada célula c_i do espaço celular é associada a um vértice v_i de G e cada aresta do grafo representa um relacionamento entre duas células c_i e c_j .

Apesar de ser um modelo matemático adequado ao estudo de fenômenos espaciais dinâmicos, os princípios básicos dos ACs são muito restritivos para serem úteis na

¹ Um espaço é isotrópico quando possui as mesmas propriedades em todas as direções (von Newman, 1996, p. 105).

modelagem de fenômenos geográficos reais, por isso algumas extensões foram propostas para esse modelo (Coullelis 1997). O espaço não deveria ser considerado homogêneo nem em suas propriedades e nem em sua estrutura, a vizinhança não precisaria ser uniforme ao longo do espaço, e as funções de transições não precisariam ser igualmente aplicadas a cada ponto do espaço. Efeitos que decaem com a distância poderiam ser mapeados nas vizinhanças do AC, as regras de transição poderiam ser probabilísticas ao invés de determinísticas e a variável tempo poderia ser usada para ajustar algum escalonamento externo. O conjunto binário de estados possíveis para uma célula, isto é, {inativo, ativo}, deveria ser estendido para um conjunto de estados locais que podem representar características de uma região como acessibilidade e restrições legais, entre outras.

As regras de transição foram generalizadas para incluir regras baseadas em modelos físicos para hidrologia (Wesseling 1996), restrições econômicas e demográficas para modelar mudanças de uso do solo (White 1997), assim como combinações de expansões baseadas em “fronteiras” e “sementes de crescimento” (Soares-Filho 2002). Em (Pedrosa et al. 2002), as regras de transição que antes estavam inseridas no escopo de um autômato finito passaram a fazer parte de um autômato híbrido (Henzinger 1996) e desta maneira, os ACs passaram a ser capazes de modelar tanto os comportamentos contínuos presentes nos sistemas reais quanto os comportamentos discretos.

CAPÍTULO 4

Requisitos de Ferramentas Computacionais para Modelagem Espacial Dinâmica

4.1 Demandas da atividade de modelagem

O processo de modelagem de fenômenos espaciais envolve as fases descritas a seguir e que não precisam acontecer na ordem em que são apresentadas, podendo uma fase ser realizada de forma concorrente com a outra. Cada uma das fases de modelagem impõe demandas diferentes.

- **Construção da base de dados:** esta etapa implica na aquisição e conversão de uma série temporal de dados espaciais para o estabelecimento de uma base de dados onde dados que permitam validar o modelo dinâmico e dados que explicam o processo de mudança são armazenados em escalas e formatos adequados. Um SIG seria a ferramenta adequada para armazenamento e tratamento desses dados. *Desta maneira, uma ferramenta para a construção modelos espaciais dinâmicos teria maior utilidade se fosse fortemente acoplada a um SIG.*
- **Parametrização do Modelo:** a parametrização do modelo consiste, essencialmente, na definição do estado inicial para a execução do modelo que é determinado pela escolha do tempo “t” a partir do qual seus estados deverão evoluir, e na definição da resolução espacial a ser utilizada ou de várias resoluções. Uma vez definida a resolução espacial, tem início o processo de agregação e alocação de dados espaciais e temporais a partir da base de dados. O termo agregação refere-se à conversão de dados a partir de subconjuntos espaciais que possuem alta resolução espacial para subconjuntos espaciais de menor resolução. O termo alocação refere-se ao processo inverso. Os resultados encontrados por (Kok e Veldkamp 2001) sugerem fortemente que qualquer esforço de modelagem deve incorporar uma

minuciosa análise dos efeitos causados pela variação da escala espacial sobre os resultados obtidos, tanto no que diz respeito à resolução espacial utilizada como a extensão da área de estudo considerada. *Portanto, uma ferramenta para a construção de modelos espaciais deveria permitir o uso de diferentes escalas espaciais na construção do modelo e fornecer serviços para agregação e alocação dos dados espaciais.*

- **Construção do Modelo:** é nesta etapa que as abordagens teórica e metodológica que influenciarão o modelo são selecionadas e utilizadas na definição das entidades que participarão do modelo e das regras que regem a sua dinâmica. Não importando se o modelo é empírico ou não, para os modelos espaciais dinâmicos existe sempre a necessidade da escolha de uma representação adequada para o tempo e para o espaço. Quando o modelo conceitual baseia-se em uma teoria, esta necessidade também se estende aos processos. De acordo com (Velkamp e Lambin 2001), é necessário que todo modelo LUCR faça distinção entre as projeções da quantidade de mudança, ou seja, demanda por mudanças, e projeções da localização das mudanças. Portanto, nesta fase o modelador também precisa definir quais modelos de demanda e de localização de mudanças serão utilizados e implementá-los em seu modelo. É indispensável que, nesta etapa, sejam modeladas as interações entre as diversas escalas do modelo e as retroalimentações existentes entre os processos e sub-sistemas do modelo. *Conseqüentemente, uma ferramenta com a que se deseja deve oferecer serviços para o modelador especifique:*

- *que dados contidos na base de dados servirão como entrada para o modelo,*
- *que dados formarão o resultado da simulação e o local onde serão gravados,*
- *os momentos no quais os resultados da simulação serão visualizados e gravados de maneira a garantir que estejam consistentes,*

- *as entidades que participarão do modelo e as regras que determinam o comportamento dessas entidades, as regras que governam o mundo que elas habitam,*
- *os mecanismos de inferência através dos quais novas regras poderão ser deduzidas,*
- *o mecanismo para cálculo da demanda por mudanças e o mecanismo utilizado para localizar as mudanças, e*
- *a ordem na qual os diversos sub-modelos devem ser executados.*

Essa ferramenta deve permitir a construção de modelos que considerem tempo e comportamentos discretos e contínuos.

- **Calibração, Verificação e Validação do Modelo:** nesta fase devem ser determinadas as variáveis que realmente possuem uma relação causa/efeito com o processo de mudança. Por exemplo, determinar se as variáveis disponibilidade de água e proximidade a estradas realmente explicam o processo de mudança do uso do solo. Após a escolha das variáveis relevantes, o modelo precisa ser calibrado através do seu ajuste aos dados disponíveis sobre a realidade. O modelo também precisa ser verificado para assegurar que sua programação e seu funcionamento estão corretos. Também é necessário que o modelo seja submetido a uma validação estrutural e uma validação do seu resultado. Isto é, uma verificação de quão perfeitamente o modelo implementado em *software* representa o modelo conceitual e uma verificação da qualidade com que o resultado do modelo caracteriza o sistema em estudo. *Portanto, uma ferramenta para construção de modelos espaciais dinâmicos deve prover serviços e métodos automatizados para a calibração, verificação e validação do modelo* (Velkamp e Lambin 2001).

- **Execução e Visualização do Modelo e Análise de Relatórios:** nesta fase, os modelos dos processos são colocados em execução sobre o espaço de células de maneira iterativa, gerando relatórios e dados espaço-temporais que registram a dinâmica do modelo para uma posterior análise. *Por conseguinte, uma ferramenta para a modelagem espacial dinâmica deve prover serviços que permitam ao modelador especificar o conteúdo do relatório desejado, que permitam a geração automática desses relatórios, e ferramentas matemáticas que permitam a análise desses e que permitam a escolha da aparência dos resultados que deverão ser visualizados.*
- **Projeção de Cenários:** nesta etapa são construídos cenários que permitem ao modelador verificar o impacto de algumas hipóteses por ele formuladas sobre o sistema modelado. Nos autômatos celulares, estas hipóteses são representadas na forma de restrições, isto é, regras definidas sobre os estados das células que têm o poder de impedir ou forçar uma determinada regra de transição. *Portanto, uma ferramenta para a modelagem espaço-temporal deve oferecer serviços que permitam ao modelador construir cenários fictícios e testar suas hipóteses.*

4.2 Demandas da ferramenta de modelagem

Uma ferramenta de modelagem espacial deve ser o mais expressiva possível. Por esse motivo, deve oferecer ao modelador inexperiente em programação formas alternativas para a construção dos modelos. Ferramentas como STELLA (<http://www.hps-inc.com>) utilizado por na arquitetura de modelagem SME (<http://www.uvm.edu/giee/SME3>), permitem ao modelador construir modelos por meio de diagramas de blocos em uma interface gráfica. A ferramenta de modelagem TerraML (Pedrosa et al. 2002) define uma linguagem de alto nível baseada em XML onde o modelador pode facilmente construir os modelos desejados.

A execução de modelos espaciais dinâmicos pode envolver a simulação de diversos modelos acoplados que interagem sobre o modelo espacial. Todos os modelos que

compartilham o espaço podem tentar, simultaneamente, alterar seus atributos o que poderia levar a condições de corrida e ao surgimento de inconsistências. Para que inconsistências sejam evitadas, todos os modelos devem compartilhar uma única base de tempo e concordar sobre a ordem na qual os eventos da simulação ocorreram. Além disso, a simulação pode consumir uma enorme quantidade de memória e poder de processamento. Portanto, uma ferramenta como a que se deseja deve ser escalável e fornecer primitivas para a comunicação e sincronização dos diversos modelos que participam da simulação.

4.3 Considerações finais

Atualmente, existe uma tendência de que os modelos espaciais dinâmicos utilizem uma abordagem híbrida entre teoria e empiricismo, sejam dinâmicos e explicitem a componente espacial, sejam capazes de considerar as diferentes escalas espaciais e suas inter-relações, utilizem modelos matemático-computacionais baseados em espaços celulares para representar o espaço, utilizem autômatos híbridos para modelar tanto processos discretos quanto contínuos e uma abordagem baseada em agente para modelar o comportamento individual de decisores e proprietários de terra. Para modelar o tempo, a tendência é de que sejam utilizadas técnicas de simulação dirigida por eventos.

Como as regras de transição que definem o comportamento dos modelos podem se basear em qualquer método matemático-computacional, uma linguagem para a construção desses modelos precisaria ser tão abrangente quanto uma linguagem de programação de uso geral. Como os fenômenos espaciais que se deseja modelar são muito complexos, o modelador teria grandes benefícios no uso das técnicas e construções encontradas no paradigma orientado por objetos e na abordagem multi-paradigma. Uma vez que teorias formais sobre as interações entre os sistemas biológicos, físicos e sócio-econômicos ainda não foram desenvolvidas (Couclelis citada em Parker et al. 2001, p. 5-6), as construções de uma linguagem de programação destinada à modelagem desses sistemas teriam grandes chances de precisarem ser alteradas conforme as pesquisas nesta área fossem avançando, o que implicaria em complexas alterações no compilador dessa linguagem.

Por estas razões, ao se projetar ferramentas computacionais para modelagem espacial dinâmica será preciso levar em conta um balanço entre diferentes requisitos:

- Dar suporte a modelos teóricos e empíricos.
- Permitir rápidas adaptações e inserção de novos modelos, com mínimas modificações no restante do ambiente computacional.
- Criar um ambiente amigável de interação entre o usuário e o ambiente de modelagem.

Na continuidade dos estudos nesta área, pretende-se estabelecer um balanço entre estes diferentes componentes (linguagens de alto nível, arcabouços de software, padrões de projeto, modelos teóricos e práticos) de formas a projetar um ambiente computacional adequado à complexidade dos problemas de modelagem espacial dinâmica.

Referências

- Alves, D. S. 2002. Space-time dynamics of deforestation in Brazilian Amazônia. **International Journal of Remote Sensing** 23: 2903-2908.
- Austern, M. 1998. **Generic Programming and the STL**. Reading, Addison-Wesley.
- Beck, K., Johnson, R. 1994. Pattern generated architecture. **European Conference on Object-Oriented Programming**, p. 139-149.
- Bennett, B. 2001. Space, time, matter and things. **Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001**.
- Berger, T., Assante, F.A., Osei-Ako, F., (in press). GLOWA-Volta Common Sampling Frame: Selection of Survey Sites. **ZEF Documentation of Research No. 1/2002**. <http://www.zef.de/zef_englisch/f_propro_neu.htm>. Visitado em 15 de setembro de 2003.
- Briassoulis, H. 2000. Analysis of Land Use Change: Theoretical and Modeling Approaches. **Tese de doutorado em geografia, Universidade de Aegean**. <<http://www.rri.wvu.edu/WebBook/Briassoulis/contents.htm>>. Vistando em 15 setembro 2003. Regional Research Institute, West Virginia University.
- Burrough, P.A., Frank, A.U. 1995. Concepts and paradigms in spatial information: are current geographical information systems truly generic? **IJGIS**, 9 (2), pp: 101-116.

- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., 1996. **Pattern-Oriented Software Architecture: A System of Patterns**. John Wiley & Sons.
- Câmara, G., Monteiro, A.M.V., Paiva, J.A.P., Gomes, J., Velho, L. 2000. Towards A Unified Framework For Spatial Data Mo Models. **Journal of the Brazilian Computing Society**, 7(1), p. 17-25.
- Câmara, G., Souza, R. C. M., et al. 2000. TerraLib: Technology in Support of GIS Inovation. **GeoInfo 2000 - II Workshop Brasileiro de Geoinformação**, São Paulo
- Coplien J. O., 1998. Software Design Patterns: Common Questions and Answers. Em Rising, L. (eds.) **The Patterns Handbook: Techniques, Strategies, and Applications**. New York, Cambridge University Press, p. 311-320.
- Coplien, J.O., 1996. Curiously Recurring Template Patterns. Em Lippman, S. (ed.) **C++ Gems**, New York, SIGS Books, p. 135-143.
- Couclelis, H., 1997. From cellular automata to urban models: new principles for model development and implementation. **Environment and Planning B**, v. 24, n. 2, p. 165-174.
- Couclelis, H., (in press). Chapter 2: Modeling frameworks, paradigms, and approaches. Em Clarke, K.C., Parks, B.E., e Crane, M.P., (2000). **Geographic Information Systems and Environmental Modelling**. New York: Longman & Co.
- Edelweiss, N., J. P. M. Oliveira 1994. **Modelagem de Aspectos Temporais de Sistemas de Informação**. Recife, UFPE-DI.
- Engenhofer, M. J., Golledge, R. G. 1998. **Spatial and temporal reasoning in geographic information systems**. New York: Oxford University Press.
- Engenhofer, M. J., Glasgow, J., Gunther, O., Herring, J., Peuquet, D. 1999. Progress in Computational Methods for Representinh Geographic Concepts, **International Journal of Geographical Information Science** 13(8): 775-796.
- Gamma, E., Helm, R. Johnson, R. Vlissides, J., 1993. Design Patterns: Abstraction and Reuse of Object-Oriented Design. **Anais do ECOOP'93**, p. 406-431.
- Gamma, E., Helm, R. Johnson, R. Vlissides, J., 1994. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison Wesley.
- Gomes, J. M., Velho, L. 1995. Abstraction Paradigms for Computer Graphics. **The Visual Computer**, 11:227-239
- Goodchild, M. 1992. Geographical Data Modelling. **Computers & Geosciences**, 18(4), 401-408.
- Goordillo, S., Balaguer, F., Mostaccio, C., Das Neves, F. 1999. Developing GIS Applications with Objects: A Design Patterns Approach. **GeoInformatica**, 3(1), 7-32.
- Henzinger, T. A., 1996. The Theory of Hybrid Automata. **Anais do 11th Symposium on Logic in Computer Science (LICS'96)**

- Hopcroft, J. E., Ullman, J. D., 1979. **Introduction to automata theory, language and computation**. Addison-Wesley Publishing Company.
- Howitt, R., e Reynaud, A., 2001. Dynamic Land Use: Data Disaggregation to Field-Level by Maximum of Entropy. **56th Econometric Society European Meeting**, Lauzane, Switzerland, 25-29.
- Jain, R., 1991. **The Art of Computer Systems Performance Analysis – Techniques for Experimental Design, Measurement, Simulation, and Modeling**. John Wiley & Sons, Inc..
- Jugurta, L.F., Cirano, L. 1999. Specifying Analysis Patterns for Geographic Databases on the Basis of a Conceptual Framework. **International Symposium on Geographic Information System**. Proceedings ACM Press. 7-13.
- Kaimowitz, D., Angelsen, A., 1998. Economic Models of Tropical Deforestation: A Review. Center for International Forestry Research.
- Kobryn, C. 2000. Modeling Components and Frameworks with UML. **Communications of the ACM**. Vol. 43, No. 10 31, p. 31-38.
- Kok, K., Veldkamp, A. 2001. Evaluating impact of spatial scales on land use pattern analysis in Central America. **Agriculture, Ecosystems and Environment** 85, p. 205-221.
- Lambin, E. F., 1994. Modeling Deforestation Processes. **A Review, Trees series B: Research Report** . European Commission, Luxembourg.
- Lambin, E.F., Baulies, X., Bockstael, N., Fischer, G., Krug, T., Leemans, R., Moran, E.F., Rindfuss, R.R., Skole, D., Turner II, B.L., Vogel, C., 1999. **Land-Use and Land-Cover Change Implementation Strategy**, IGBP Report No. 48/IHDP Report No. 10, IGBP, Stockholm, 125 pp.
- Lambin, E.F., Rounsevell, M., Geist, H., 2000. Are current agricultural land use models able to predict changes in land use intensity? **Agriculture Ecosystems and Environment**, 1653, 1–11.
- Laszlo, E. e Margenau, H., 1972. The emergence of integrating concepts in contemporary science. **Philos. Sci.** 39:252-259.
- Lee, E. A., Varaiya, P., 2003. **Structure and Interpretation of Signals and Systems**. Addison-Wesley.
- Lim, K., Deadman, P. J., Moran, E., Brondízio, E., McCracken, S. 2002. Agent-Based Simulations of Household Decision Making and Land Use Change near Altamira, Brazil. **Integrating Geographic Information Systems and Agent-Based Modeling: Techniques for Simulating Social and Ecological Processes**, ed. R. Gimblett, p. 277–310. Santa Fe Institute Studies in the Sciences of Complexity series. New York: Oxford University Press.
- Minsky, L. M., 1967. **Computation: finite and infinite machines**. Prentice-Hall, Inc..

- O'Sullivan, D., 2001. Graph-cellular automata: a generalised discrete urban and regional model. **Environment and Planning B: Planning and Design**. 28: p. 687-705.
- Parker, D.C., Berger, T., Manson, S. M., 2001. Agent-Based Models of Land-Use and Land-Cover Change, LUCC Report No.6, **Report and Review of an International Workshop**, Irvine, California, USA.
- Parent, C., Spaccapietra, S., Zimányi, E. 1999. Spatio-temporal conceptual models: data structures + space + time. **Anais do seventh ACM international symposium on Advances in geographic information systems**.
- Parks, B. O. 1993. The Need for Integration. **Environmental Modelling with GIS**, edited by M. J. Goodchild, B. O. Parks and L. T. Steyaert. Oxford: Oxford University Press.
- Peuquet, D. 2001. Making Space for Time: Issues in Space-Time Data Representation. **GeoInformatica** 5(1): 11-32.
- Pedrosa, B. , Câmara, G. Fonseca, F., Carneiro, T. Souza, R. 2002. TerraML: a Language to Support Spatial Dynamic Modeling. **Anais do Congresso Brasileiro de GeoInformação**.
- Pfoser, D., Tryfona, N. 1998. Requirements, definitions, and notations for spatiotemporal application environments. **Anais do sixth ACM international symposium on Advances in geographic information systems**.
- Santos, Milton. 1996. **A Natureza do Espaço (The Nature of Space)**. São Paulo: Hucitec.
- Reis, E. J., Margulis, S. 1991. Options for Slowing Amazon Jungle Clearing. **Global warming: economic policy responses**. R. Dornbusch and J. M. Poterba. Cambridge, The MIT Press: 335-375.
- Russel, S. J, Norvig P., 1995. **Artificial Intelligence – A Modern Approach**. Prentice Hall. Nova Jersey.
- Schmidt, D.C., Fayad, M., Johnson, R. 1996, Software Pattern. **Communications of the ACM**, Vol. 39, No. 10, p. 37-39.
- Smyth, C. S., 1998. A representational framework for geographic modeling. Em **Spatial and temporal reasoning in geographic information systems**. Eds. Max J. Egenhofer e Reginald G. Golledge, 1991-213. New York: Oxford University Press.
- Soares Filho, B. S. 1998. Modelagem dinâmica de paisagem de uma região de fronteira de colonização amazônica. **Tese de doutorado, Escola Politécnica**. São Paulo, Universidade de São Paulo.
- Soares-Filho, B. S., G. C. Cerqueira, et al., 2002. DINAMICA: A New Model to Simulate and Study Landscape Dynamics. **Ecological Modelling**.
- Sipper, M. 1999. The emergence of Cellular Computing. **IEEE Computer** 32(7): 18-26.

- Takeyama, M.; Couclelis, H. Map dynamics: integrating cellular automata and GIS through geo-algebra. **International Journal of Geographical Information Science**, v. 11, p. 73-91, 1997.
- Tomlin, C.D., 1990. **Geographic Information Systems and Cartographic Modeling**. Prentice Hall.
- Tryfona, N., Jensen, C. S. 2000. Using abstractions for spatio-temporal conceptual modeling. **Anais do 2000 ACM Symposium on Applied Computing**.
- Turner II, B.L., W.B. Meyer and D.L. Skole. 1994. Global Land-Use/Land-Cover Change: Towards an Integrated Program of Study. **Ambio** 23 (1): 91-95.
- Turner, B.L. II and B.L. Meyer. 1994. Global Land Use and Land Cover Change: An Overview. Em **Changes in Land Use and Land Cover: A Global Perspective**, eds. W.B. Meyer and B.L. Turner II, 3-10. Cambridge: Cambridge University Press.
- Turner, B.L. II, Skole, S., Saderson G., Fischer, L., Fresco, L., Leemans, R., 1995. Land-Use and Land-Cover Change Science/Research Plan. **IGBP Report No.35 and IHDP Report No. 7**. Stockholm: International Geosphere-Biosphere Programme (IGBP); Geneva: International Human Dimensions Programme on Global Environmental Change (IHDP).
- UML Revision Task Force 1999. **OMG Unified Modeling Language Specification, v. 1.3, document ad/99-06-08**. Object Management Group.
- Veldkamp, A., Fresco, L. O. 1996. CLUE: a conceptual model to study the Conversion of Land Use and its Effects. **Ecological Modelling** 85, p. 253-270.
- Veldkamp, A., Lambin, E.F., 2001. Predicting land-use Change. **Agriculture Ecosystems and Environment**. 85, 1-6.
- Verburg, P. H., Soepboer, W., Veldkamp, A., Limpiada, R., Espaldon, V., Mastura, S. S. A., 2002. Modeling the Spatial Dynamics of Regional Land Use: The CLUE-S Model. **Environmental Management**, Vol. 30, No. 3, p. 391-405.
- von Neumann, J., 1966. **Theory of Self-Reproducing Automata**. University of Illinois Press, Illinois. Edited and completed by A.W. Burks.
- Wesseling, C.G, D. Karssenbergh, W.P.A Van Deursen, and P.A Burrough, 1996. Integrating dynamic environmental models in GIS: the development of a Dynamic Modelling language. **Transactions in GIS**, 1:40-48.
- Wiener, N. 1948. **Cybernetics**. (Segunda edição, 1961.) Cambridge, Mass., MIT Press.
- Wilson, A. G., 1981. **Geography and the Environment Systems Analytical Methods**, Chichester: Jhon Wiley & Sons, Ltd.
- White, R. W.; Engelen, G., 1997. Cellular automaton as the basis of integrated dynamic regional modelling. **Environmental and Planning B**, v.24, p.235-246.

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

Wolfram, S. 1983. Cellular Automata. **Los Alamos Science**, No. 9, p. 2-21.

Wolfram, S., 1984 . Cellular automata as models of complexity. **Nature**, No. 311, p. 419-424.