Disciplina Engenharia de Software I – BCC322

Trabalho Final: Parte 1 – Desenvolvimento de um Framework C++ para a Construção de Simulações Baseadas na Dinâmica de Sistemas.

Prof. Tiago Garcia de Senna Carneiro

Muitas as áreas da ciência moderna organizam o conhecimento sobre seu objeto de estudo de uma mesma forma modular e hierárquica, onde partes observáveis do mundo real são chamadas "sistema" (módulos) e são descritas como agregações de outros sistemas que são, portanto, chamados subsistemas (relação hierárquica, parentesco). Por exemplo, na medicina, o corpo humano é dividido em sistema cardiovascular, sistema reprodutor, etc. O sistema cardiovascular é formado pelo coração, pulmão, etc. O coração, por sua vez, é um sistema formado pelo ventrículo direito, ventrículo esquerdo, etc. Na mecânica de automóveis, o veículo é dividido em sistema de frenagem, sistema de arrefecimento, etc. Cada um desses sistemas também pode ser dividido em subsistemas. Na cosmologia, o sistema solar, possui em seu interior o sistema Terrestre. Na computação, sistemas de computação são formados pela interconexão de vários outros sistemas. Na Ecologia, os ecossistemas podem ser terrestres, aquáticos, etc. No Direito, a Constituição é um sistema de normas que fornece um controle geral aos sistemas jurídicos: do direito privado, do direito tributário, do direito penal, etc. Este fato se deve as idéias que povoavam as mentes de intelectuais e cientistas no pós segunda guerra mundial, e foram formalizadas pelo biólogo austríaco Ludwig von Bertalanffy (1950), no seu melhor trabalho "Teoria Geral de Sistemas".

Segundo a teoria de sistemas, que estuda a representação abstrata de fenômenos independente de sua natureza, sistemas formados por elementos de mesma função e que interagem da mesma forma com os outros elementos do sistema exibirão um mesmo comportamento, independentemente, da natureza desses sistemas. Por exemplo, se duas diferentes populações de animais apresentam a mesma organização e funcionamento como, por exemplo, apenas um único casal em toda a população procria e lidera o grupo, então não importa que animais sejam estes, suas populações apresentarão exatamente a mesma dinâmica.

Seguindo os preceitos da Teoria Geral de Sistemas, J. W. Forrester (1968) desenvolveu uma linguagem para a descrição de sistemas dinâmicos, onde sistemas são descritos como variáveis numéricas que representam o estoque de certa quantidade de energia ou matéria, e as interações entre os sistemas são representadas por fluxos que carregam energia ou matéria de um sistema para outro. Quando representada graficamente, como na figura 1, a linguagem DYNAMO utiliza um retângulo para denotar sistemas e uma seta para denotar fluxos, onde a direção da seta indica o sentido do fluxo (da origem para o destino).

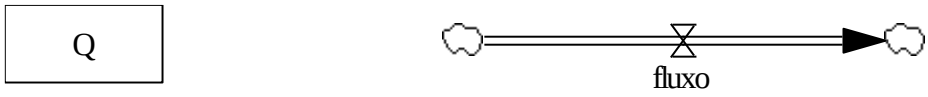

Figura 1. O sistema "Q" é denotado graficamente como um retângulo e representa o estoque de qualquer quantidade de energia ou matéria. A seta representa um fluxo que transporta energia ou matéria de sua origem para o destino.

Enquanto sistemas são computacionalmente representados por uma variável numérica (acumulador), fluxos são descritos por equações algébricas. Por exemplo, o modelo na figura 2 (a) simula uma população que cresce a uma taxa de natalidade constante igual a 30% e, na figura 2 (b), a hipótese de que a população não pode crescer indefinidamente é adicionada ao sistema, por meio do termo (1 – P/Pmax) inserido na equação do fluxo de natalidade.

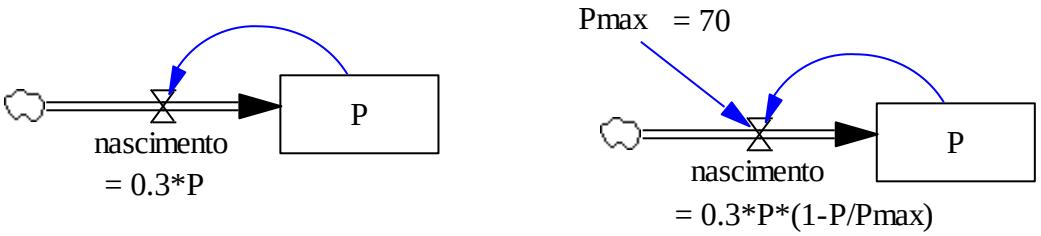

Figura 2. Modelos de crescimento exponencial (a) e logístico (b)

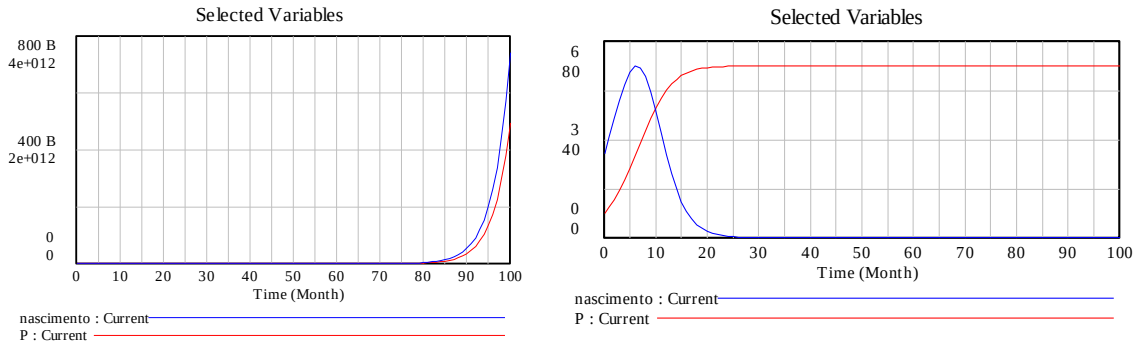

Figura 3. Resultados dos modelos exponencial (a) e logístico (b) para uma população inicial igual a 10.

A figura 3 apresenta o resultado da execução dos modelos apresentados na figura 2 para uma população inicial igual a 10 indivíduos, e um período de tempo igual a 100 anos. O modelo da figura 2 (a) prevê o crescimento exponencial da população P, baseado na hipótese de que os recursos necessários para manter a população viva e em crescimento são infindáveis. O modelo da figura 2(b) prevê um crescimento logístico para a população, na qual ela não cresce a uma taxa constante no tempo, conforme a população cresce e se aproxima de um limite máximo, Pmax, definida pela quantidade

de recursos disponíveis para aquela população, o fluxo de nascimento reduz sua intensidade devido ao termo (1 – P/Pmax). Pmax é a capacidade máxima de suporte do ambiente com relação à população nele inserida.

A linguagem DYNAMO foi implementada em um simulador de mesmo nome, cuja estrutura principal era idêntica a de um simulador discreto, na qual os modelos a simulados eram executados internamente ao loop principal do simulador, parametrizado pelos tempos de inicio e fim da simulação, respectivamente, t_incial e t_final. Veja na figura 4 o loop principal de simulação dos modelos apresentados na figura 2.

```
// PARAMETROS DO MODELO
P = 10; // população incial

// LOOP PRINCIPAL
for( tempo = t_inicial; tempo <= t_final; tempo ++)
{
        nascimento = 0.3* P;
        P = P + nascimento;
        cout << P;
}
```

```
// PARAMETROS DO MODELO
P = 10; // população incial
Pmax = 70; // população máxima

// LOOP PRINCIPAL
for( tempo = t_inicial; tempo <= t_final; tempo ++)
{
        nascimento = 0.3* P*(1 – P/Pmax);
        P = P + nascimento;
        cout << P;
}
```

Figura 4. Simuladores discretos para os modelos de crescimento exponencial (a) e logístico (b).

```
// PARAMETROS DO MODELO
sistema[1] = valorInicial[1];
sistema[2] = valorInicial[2];
...
Sistema[n] = valorInicial[n];

// LOOP PRINCIPAL
for( tempo = t_inicial; tempo <= t_final; tempo ++)
{
        para cada fluxo “i” pertencente ao modelo   {   v[i] = fluxo[i].executa( ); }

        para cada fluxo “i” pertencente ao modelo
        {
                sistemaOrigem = fluxo[i].getSistemaOrigem( )
                sistemaOrigem = sistemaOrigem - v[i];

                sistemaDestino = fluxo_i.getSistemaDestino( )
                sistemaDestino = sistemaDestino + v[i]

        }
}

// RELATA RESULTADOS
para cada sistema “i” pertencente ao modelo{  imprime( sistema[i] ) }
```

Figura 5. Estrutura geral de um algoritmo geral para a simulação de modelos baseados na Teoria Geral de Sistemas.

Analisando-se os códigos apresentados na figura 4 é possível a identificação de uma estrutura comum e a definição de um algoritmo geral para a simulação de modelos baseados na Teoria de Sistemas, como mostra a figura 5, onde as funções “executa( )” dos fluxos "fluxo[i]" dependem das hipóteses do modelo e devem ser definidas pelo

modelador com base em seu conhecimento sobre o domínio da aplicação. Durante a simulação, a cada instante de tempo, todos os fluxos definidos no modelo são simulados sequencialmente e o valor resultante é armazenado temporariamente. Depois, o simulador percorre novamente os fluxos subtraindo o respectivo valor a seu sistema origem e somando a seu sistema destino. Desta maneira, a quantidade de energia armazenada em cada sistema é atualizada. Finalmente, o simulador relata esta quantidade estocado no sistemas a seu usuário.

QUESTOES:

Concepção e Projeto da API em C++

1) Identifique os "casos de uso" aos quais sua API deverá satisfazer. Então, identifique os "critérios de aceitação" do cliente. Dica: Baixe o software "Vensim PLE", abra o arquivo "ValidacaoMyVensim.mdl" e o execute. Assim, os "critérios de aceitação" poderão de identificados.

2) Valide os "casos de uso" e "critérios de aceitação" com o cliente (papel interpretado pelo professor).

3) Para projetar sua API, estude diversas alternativas de uso dessa API na implementação dos modelos que configuram os "casos de uso" validados pelo cliente. Este estudos, devem dar origem a "cenários de testes" que verificam os diversos "casos de uso" e "critérios de aceitação". Finalmente, apresente os pseudo-códigos materializam os "cenários de teste".

4) Utilize a notação UML (diagramas de classe) para especificar a API (Application Programming Interface) de uma biblioteca C++ destinada ao desenvolvimento de modelos ambientais baseados na Teoria de Sistemas.

Construção e Teste

5) Implemente, em C++, os "cenários de testes" na forma de "testes funcionais" automatizados que permitam verificar o funcionamento da API desenvolvida.

6) Implemente, em C++, a API especificada na questão 1 utilizando C++ e as melhores práticas de programação vistas nas listas de exercícios, slides e em sala de aula.

7) Repita esses passos até que todos os "critérios de aceitação" sejam validados pelo cliente e verificados/homologados por meio de "testes funcionais" automatizados.