UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CIÊNCIAS DA COMPUTAÇÃO

João Fellipe Uller (17102812)

Projeto Prático: PP2 – Modelo de simulação de transmissão confiável com sliding window

INE5429 – Modelagem e Simulação
Professor: Rafael Luiz Cancian

Florianópolis
2019/2

1. Introdução

Esta categoria de projetos práticos envolve reproduzir o comportamento de um modelo de simulação discreta orientada a eventos (criado para o simulador "Arena Simulation") no simulador "Genesys", do professor Cancian. Esses projetos envolvem inicialmente executar no Arena um modelo existente, pronto e funcional que descreve certo sistema (descrição a seguir), e coletar resultados estatísticos de sua execução. Depois, implementar todos os componentes e elementos análogos (e qualquer outra infraestrutura faltante, como geração de distribuições de probabilidade ou estatísticas) no simulador Genesys, criar um modelo equivalente no simulador Genesys, executá-lo nesse simulador e chegar a resultados estatisticamente equivalentes (comprovados por testes de hipóteses).

1.1 Enunciado do projeto

O desenvolvedor de um sistema operacional deseja avaliar o impacto

- (a) do tamanho das janelas deslizantes;
- (b) do tempo máximo do timer para recebimento do ack;
- (c) da taxa de erro de transmissão sobre o desempenho do sistema de comunicação da camada de transporte do protocolo TCP/IP, que é medido como o a taxa média de transmissão por segundo.

O sistema é composto por um nodo transmissor e um nodo receptor, além do canal de comunicação. De modo geral, o nodo transmissor envia um novo pacote assim que receber a confirmação de recebimento do pacote anterior. O tamanho dos pacotes (em bytes) foi amostrado muitas vezes e os dados estão no arquivo "tamanho_pacote_bytes.txt". Cada pacote transmitido segue pelo canal de comunicação, cuja taxa de transferência é de 100Mb/s. Nesse canal de comunicação os pacotes seguem exatamente a mesma rota (entrega ordenada), sendo que o tempo de propagação do canal pode variar de 1us a 50us (níveis mínimo e máximo). A taxa de erro de comunicação (transmissão ou recepção) é tal que a probabilidade de um pacote conter algum erro pode variar de 0,01% a 5% (níveis mínimo e máximo).

De forma geral, para garantir a entrega correta dos pacotes, após enviar um pacote, o transmissor liga um timer que estabelece o prazo máximo no qual o transmissor irá esperar pelo recebimento de um pacote de acknowledge (ack), indicando que aquele pacote foi recebido sem erros pelo receptor. O timer pode ser configurado de 75us até 2ms (níveis mínimo e máximo).

Se o transmissor não receber o pacote de ack até o limite do timer, ele reenvia o pacote. Se receber o ack, o transmissor entende que o pacote foi entregue com sucesso e pode enviar o próximo pacote. Se para cada pacote transmitido o transmissor esperar por um ack, tem-se o que se chama de protocolo stop-and-wait, ou janela deslizante de tamanho n=1, o que costuma ser ineficiente. O sistema a ser modelado utiliza o protocolo de janelas deslizantes com tamanho n>1 e go-back-n (n varia de 5 a 50, níveis mínimo e máximo). Nesse protocolo, o transmissor possui uma "janela" de n quadros, o que significa que ele pode enviar n quadros consecutivos antes de ter que parar e esperar por um ack. Sempre que o ack de um quadro é

recebido, o transmissor pode enviar o próximo. Se o ack de um quadro específico não for recebido até o disparo do timer, o transmissor reinicia a janela a partir daí, retransmitindo todos os quadros a partir daquele cujo ack não foi recebido.

O receptor sabe qual é o número sequencial do próximo quadro que ele espera e, ao recebê-lo, se ele estiver correto (sem erros), envia um quadro de ack de volta ao remetente. O quadro de ack possui 128 bytes. Se o receptor receber um quadro com erro ou um quadro que não é o próximo da sequência (aquele que ele espera), simplesmente ignora o quadro.

Além de avaliar o impacto dos fatores citados sobre a métrica de desempenho, o desenvolvedor do sistema operacional deseja obter estatísticas sobre a taxa de pacotes retransmitidos e sobre a taxa efetiva de transmissão.

1.2 Objetivos

1.2.1 Objetivos gerais

- Coletar os dados gerados pelo modelo existente no Arena;
- Implementar os componentes e elementos análogos necessários ao modelo que ainda não foram desenvolvidos no simulador Genesys;
- Testar e avaliar o correto funcionamento dos componentes já implementados;
- Criar um modelo equivalente e executá-lo no simulador Genesys;
- Obter no Genesys resultados estatisticamente equivalentes àqueles coletados com o modelo original no simulador Arena.

1.2.2 Objetivos específicos

- 1.2.2.1 : implementar e testar o Hold;
- 1.2.2.2 : implementar e testar o Signal;
- 1.2.2.3 : testar o componente já implementado Separate;
- 1.2.2.4 : testar o componente já implementado Remove;

1.3 Etapas e Desenvolvimento

Tem-se a seguir uma breve descrição de como deverá acontecer a realização de cada uma das etapas necessárias para que se atinjam os objetivos previamente definidos.

1.3.1 Coleta de dados no simulador Arena (1 h)

Nesta primeira etapa, serão coletados os dados e as estatísticas geradas pela simulação do modelo já existente, previamente descrito no enunciado, em uma versão *Student* do *Arena Simulation Software*. Esta coleta de dados será realizada de modo a obter a base para os testes de hipótese que mais adiante avaliarão a corretude do modelo a ser criado no simulador Genesys e nos resultados extraídos do mesmo.

1.3.2 Teste e implementação dos componentes Hold e Signal (6 h)

Nessa etapa serão implementados os componentes *hold* e *signal* que geralmente trabalham em conjunto na composição de modelos de simulação, por isso serão implementados e testados em uma mesma etapa.

1.3.3 Teste do componente Separate (2 h)

Nessa etapa será desenvolvido o teste e a avaliação do funcionamento correto do componente *separate*, previamente desenvolvido em semestres anteriores, além de possíveis correções para que atinja o funcionamento desejado.

1.3.4 Teste do componente Remove (2 h)

O último componente a ser testado será o *remove*, previamente desenvolvido em semestres anteriores. Este também terá um teste unitário e possíveis correções para seu correto funcionamento serão realizadas nessa etapa.

1.3.5 Criação e execução do modelo no simulador Genesys (4 h)

Uma vez que os componentes necessários para a construção do modelo estejam implementados no Genesys, será realizada a construção do modelo em si, seguindo a mesma estrutura do modelo original no Arena, de modo a reproduzir esse comportamento agora no Genesys.

1.3.6 Obtenção de dados equivalentes no simulador Genesys (8 h)

Por fim, a execução do modelo criado será realizada agora no Genesys e a avaliação dos resultados obtidos será realizada comparando as saídas com aquelas obtidas em 1.3.1 no simulador Arena, checando se essas respostas são estatisticamente equivalentes e atestando, assim, a corretude ou não do modelo criado no Genesys.

1.4 Testes

1.4.1 Testes unitários

Testes unitários realizados para testar cada componente desenvolvido individualmente, de modo a atestar o seu correto funcionamento antes de inseri-los nos modelos a serem simulados.

1.4.2 Simulação de modelos simplificados (Testes de integração)

Construção de modelos simplificados para verificar a corretude no comportamentos de blocos de componentes atuando em conjunto, atestando seu funcionamento quando integrados com os demais componentes desenvolvidos. Serão utilizados, principalmente, para análise e depuração sobre quais blocos estão tendo comportamentos discrepantes em relação aos mesmos blocos do modelo original.

1.4.3 Testes de validação (Testes de hipótese)

Testes de hipótese para avaliar estatisticamente a equivalência do modelo criado no Genesys ao modelo original no Arena. Essa análise comparativa dos resultados obtidos com os dois modelos será feita de modo a comprovar se o modelo criado no Genesys reproduz, de fato, o mesmo comportamento simulado pelo modelo original no Arena.

1.5 Cronograma

Início do projeto: 21/08

- Entrega Parcial I (EP1): 26/08
- Entrega Parcial II (EP2): 09/09
- Entrega Parcial III (EP3): 23/09
- Entrega Parcial IV (EP4): 07/10
- Entrega Parcial V (EP5): 21/10

Fim do projeto: 04/11 – Entrega Final (EF)

18/11 – Prazo para ajustes (EA)

Semana / Atividade	**1 EP1**	2	**3 EP2**	4	**5 EP3**	6	**7 EP4**	8	**9 EP5**	10	**11 EF**	12	**13 EA**
1.3.1	X												
1.3.2		X	X	X									
1.3.3				X	X								
1.3.4					X	X							
1.3.5						X	X	X					
1.3.6								X	X	X	X		
Ajustes												X	X

Tabela 1 – Diagrama de Gantt – Distribuição das atividades ao longo do tempo

1.6 Repositório GitHub

Link para o repositório: https://github.com/joaofel-u/pp2-ine5425

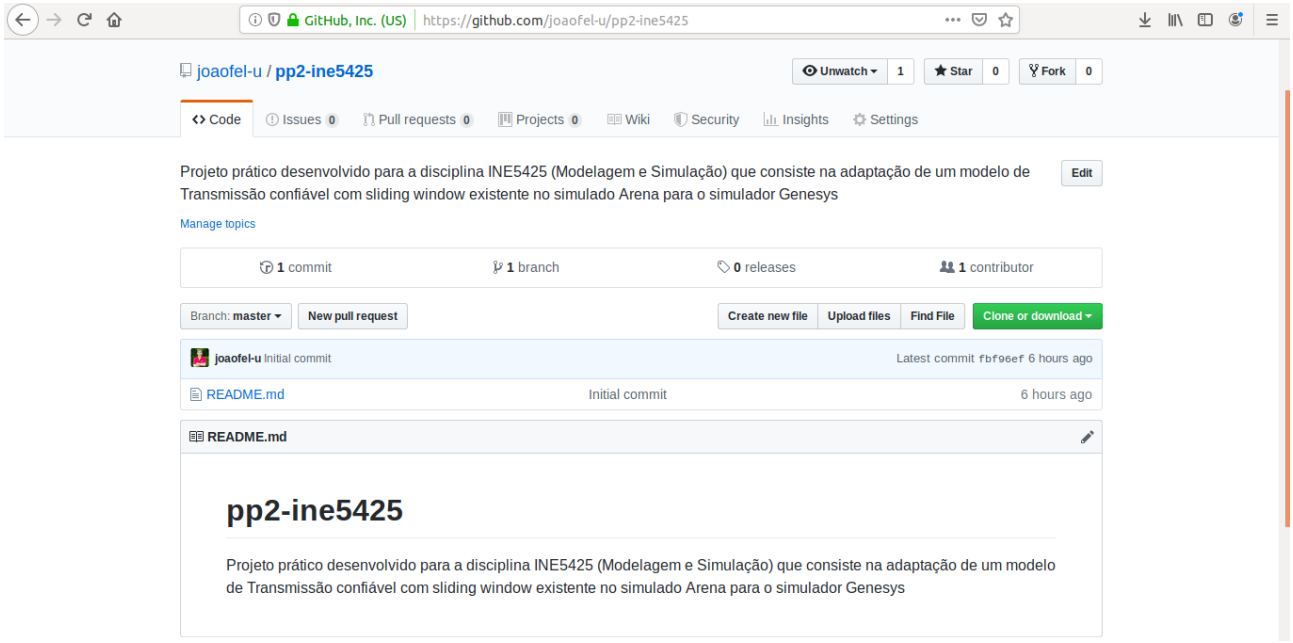


Figura 1 – Print-screen do repositório utilizado no GitHub para desenvolvimento do projeto.

2. Desenvolvimentos

2.1 Remove.cpp

Para esta entrega foi finalizada a implementação do componente Remove, tendo sido realizados os ajustes sobre os arquivos desenvolvidos por alunos do semestre passado após a aplicação de testes unitários.

2.2 Separate.cpp

Para essa entrega, o Separate foi implementado pelo estudante Ad Nunes, sendo então reaproveitada essa implementação neste trabalho. Foram realizados algumas modificações nesse arquivo em relação ao disponibilizado, principalmente com relação ao funcionamento quando em modo de Duplicação, restando ainda no entanto implementar a divisão também dos atributos com base em *CostToDuplicate*, ao invés de simplesmente copiar a entidade original várias vezes.

2.3 Hold.cpp

O componente Hold foi aproveitado da implementação do estudante Gustavo Borges, no entanto devido à disponibilização tardia, e ele ter vindo com várias funcionalidades faltando, fazem-se necessários mais alguns ajustes, no entanto uma versão funcional para o modelo já se faz disponível.

2.4Signal.cpp

O Signal, assim como o Hold, foi disponibilizado por Gustavo Borges, mas da mesma maneira não aparece completamente funcional. Porém avanços em relação à ultima entrega já foram feitos e o mesmo está próximo de ser utilizável.

2.5 SlidingWindow.cpp

Nesta entrega o modelo está completamente construído e pronto para se avaliar os seus resultados, uma vez que todos componentes estejam devidamente finalizados, restando apenas a adequação do modelo aos dados obtidos pela execução do modelo original no Arena.

2.6 ModelCheckerImpl2.cpp

Implementação da interface ModelChecker que permite a execução de modelos com mais de 50 componentes, que é o caso do modelo em questão.

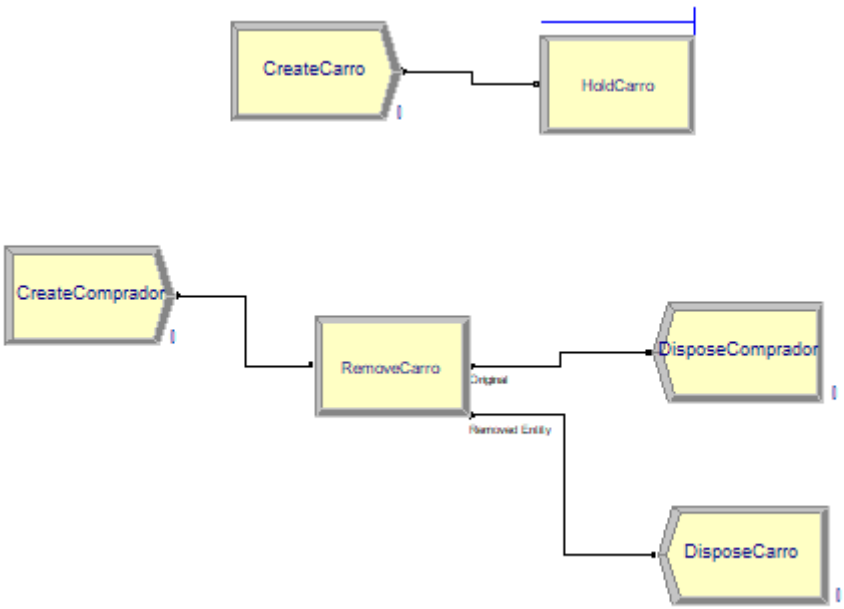
2.7 PluginConnectorDummyImpl2.cpp

Implementação da interface PluginConnector para inserção e conexão de todos os componentes necessários ao modelo no Genesys. Criado para que não se precise alterar o arquivo antes utilizado (PluginConnectorDummyImpl1.cpp).

3. Testes

3.1 TestRemove.cpp

Para os testes do componente Remove, foi criado um modelo simplificado no Genesys, equivalente ao mostrado a seguir:

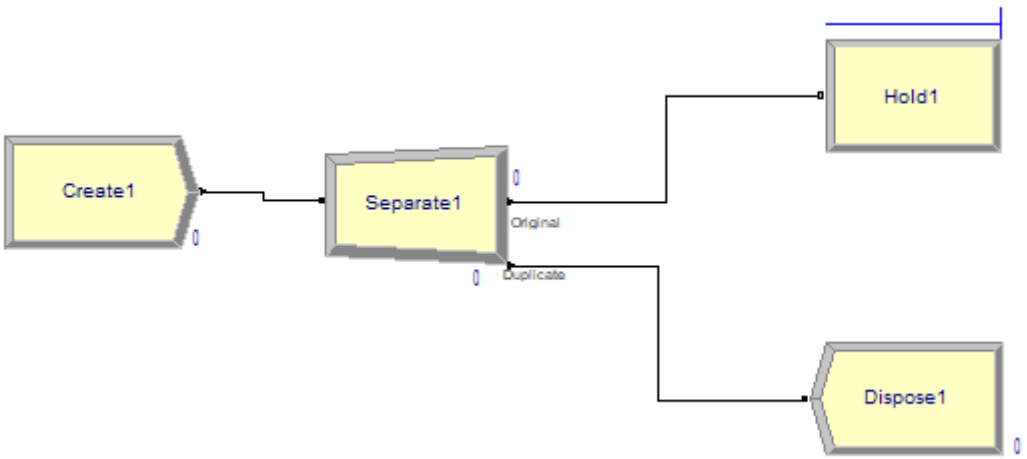


```
Replication 1 of 1 is starting.
|  Processing event=(time=0.000000,entity=1,comp="CreateCarro")
|  |  Entity 1 has arrived at component "CreateCarro"
|  |  Entity 1 has arrived at component "HoldCarro"
|  Processing event=(time=5.000000,entity=2,comp="CreateComprador")
|  |  Entity 2 has arrived at component "CreateComprador"
|  |  Entity 2 has arrived at component "RemoveCarro"
|  |  Entity 1 has arrived at component "DisposeCarro"
|  |  Entity 2 has arrived at component "DisposeComprador"
```

```
Begin of Report for replication 1 of 1
|  Statistis for Create:
|  |  CreateCarro:
|  |  |  name                                  elems
|  |  |  Count number in..........................  13
|  |  CreateComprador:
|  |  |  name                                  elems
|  |  |  Count number in..........................  6
|  Statistis for Dispose:
|  |  DisposeCarro:
|  |  |  name                                  elems
|  |  |  Dispose_2.Count_number_out..............  6
|  |  DisposeComprador:
|  |  |  name                                  elems
|  |  |  Dispose_1.Count_number_out..............  6
```

3.2 TestSeparate.cpp

Para os testes do Remove, o modelo simplificado no Genesys é equivalente ao mostrado a seguir:




```
Replication 1 of 1 is starting.
|  Processing event=(time=0.000000,entity=1,comp="Create1")
|  |  Entity 1 has arrived at component "Create1"
|  |  Entity 1 has arrived at component "Separate1"
|  |  Entity 1 has arrived at component "Hold1"
|  |  Entity 3 has arrived at component "Dispose1"
|  Processing event=(time=10.000000,entity=2,comp="Create1")
|  |  Entity 2 has arrived at component "Create1"
|  |  Entity 2 has arrived at component "Separate1"
|  |  Entity 2 has arrived at component "Hold1"
|  |  Entity 5 has arrived at component "Dispose1"
```

3.3 TestHoldSignal.cpp

Para os testes dos componentes Hold e Signal, o modelo simplificado no Genesys é equivalente ao mostrado a seguir: