

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

- Coletar os dados gerados pelo modelo existente no simulador Arena;
- Implementar os componentes e elementos análogos que ainda não foram desenvolvidos, necessários ao modelo, 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.

1.3 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 correteza do modelo a ser criado no simulador Genesys e nos resultados extraídos do mesmo.

1.3.2 Teste / implementação dos componentes necessários no simulador Genesys

Nesta etapa do desenvolvimento, serão desenvolvidos os componentes e elementos, análogos àqueles criados no Arena, necessários para a criação do modelo no Genesys e que ainda não foram implementados neste. Para os que já foram implementados, será avaliado e testado se os mesmos se comportam da maneira esperada e cumprem com aquilo que se espera destes.

Alguns dos componentes/elementos que deverão ser implementados nesta etapa são:

- 1.3.2.1 : implementar Hold (3 h);
- 1.3.2.2 : implementar Signal (3 h);
- 1.3.2.3 : testar Separate (2 h);
- 1.3.2.4 : testar Remove (2 h);

1.3.3 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.

1.3.4 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 sempre 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) (4 h)

Construção de modelos simplificados do modelo original como uma forma de verificar a corretude dos comportamentos de blocos maiores de componentes atuando em conjunto, atestando assim seu funcionamento quando trabalhando em conjunto com os demais componentes desenvolvidos.

1.4.3 Testes de validação (Testes de hipótese)

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

1.5 Cronograma

21/08 – Definição do tema e início do projeto

26/08 – Entrega Parcial I

- Proposta inicial do projeto

09/09 – Entrega Parcial II

- Item 1.3.1 (Coleta dos dados do modelo original no Arena);
- Item 1.3.2.1 (Hold);
- Item 1.3.2.2 (Signal);

23/09 – Entrega Parcial III

- Item 1.3.2.3 (testar Separate);
- Item 1.3.2.4 (testar Remove);
- Item 1.3.3 (Criação e execução do modelo no Genesys)

07/10 – Entrega Parcial IV

- Item 1.3.4 (Obtenção de resultados equivalentes no Genesys)

21/10 – Entrega Parcial V

- Item 1.3.4 (Obtenção de resultados equivalentes no Genesys)
- Item 1.4.3 (Testes de validação)

04/11 – Entrega Final

18/11 – Ajustes Finais

1.6 Repositório GitHub

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

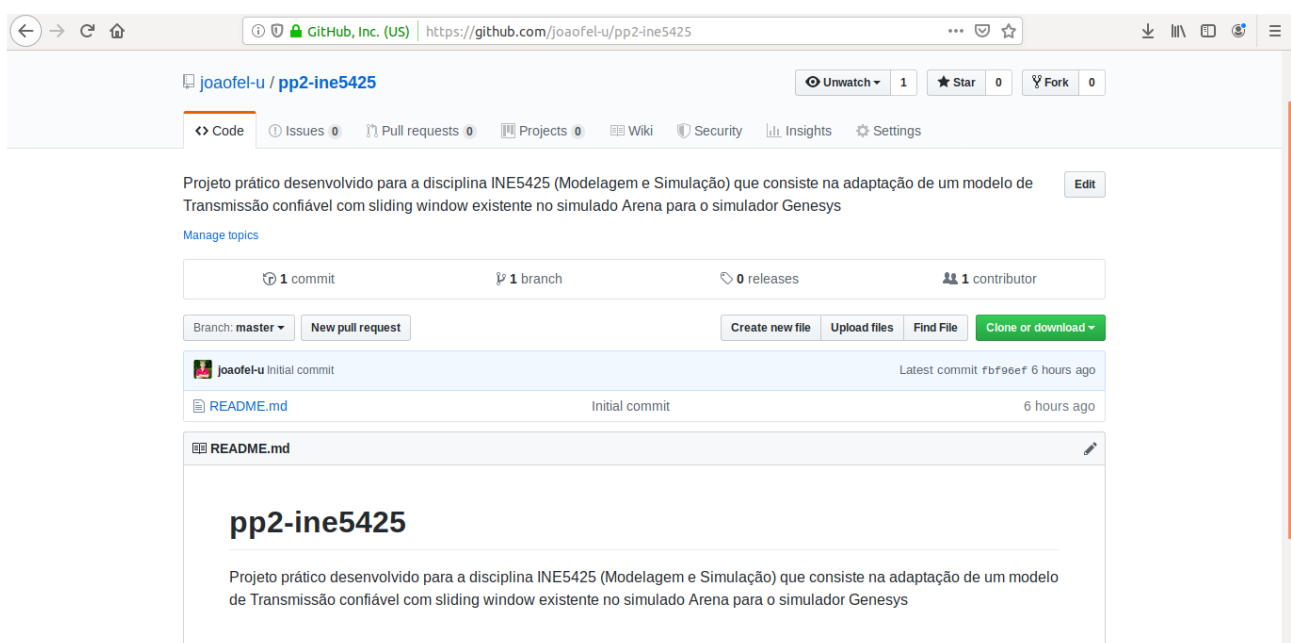


Figura 1 – Print-screen do repositório utilizado no GitHub para desenvolvimento do projeto.

2. Desenvolvidimentos

2.1 Hold.h

```
21 class Hold : public ModelComponent {
22
23 public:
24
25     enum class Type: int {
26         WaitForSignal=0,
27         ScanForCondition=1,
28         InfiniteHold=2
29     };
30
31     Hold(Model* model);
32     Hold(const Hold& orig);
33     virtual ~Hold();
34
35     virtual std::string show();
36     static PluginInformation* GetPluginInformation();
37     static ModelComponent* LoadInstance(Model* model, std::map<std::string, std::string>* fields);
38
39     void setWaitForValueExpr(std::string _expr);
40     void setLimit(std::string _limit);
41     void setCondition(std::string _condition);
42     void setType(Type _type);
43     void setQueue(Queue* _queue);
44
45     std::string getWaitForValueExpr() const;
46     std::string getLimit() const;
47     std::string getCondition() const;
48     Type getType() const;
49     Queue* getQueue() const;
50
51     void handleSignalReceived(int sigVal, int sigLimit);
52
53 protected:
54     virtual void _execute(Entity* entity);
55     virtual bool _loadInstance(std::map<std::string, std::string>* fields);
56     virtual void _initBetweenReplications();
57     virtual std::map<std::string, std::string>* _saveInstance();
58     virtual bool _check(std::string* errorMessage);
59
60 private:
61     void releaseEntity(Entity* entity, double startedWaiting);
62
63 private:
64     std::string _waitForValue = "1.0";
65     std::string _limit = "0.0";
66     std::string _condition = "";
67     Type _type = Type::WaitForSignal;
68     Queue* _queue;
```

Figura2 – Hold.h

2.2 Hold.cpp

```
141 void Hold::_execute(Entity* entity) {
142     if (this->_type == Type::WaitForSignal || this->_type == Type::InfiniteHold) {
143         Waiting* waiting = new Waiting(entity, this, _model->getSimulation()->getSimulatedTime());
144         this->_queue->insertElement(waiting);
145         _model->getTraceManager()->traceSimulation(Util::TraceLevel::blockInternal, _model->getSimulation()->getSimulatedTime(), entity,
146     } else {
147         // IMPLEMENT CORRECTLY
148         Waiting* waiting = new Waiting(entity, this, _model->getSimulation()->getSimulatedTime());
149         this->_queue->insertElement(waiting);
150         double condition = _model->parseExpression(_condition);
151         _model->getTraceManager()->traceSimulation(Util::TraceLevel::blockInternal, _model->getSimulation()->getSimulatedTime(),
152         if (condition)
153             _model->sendEntityToComponent(entity, this->getNextComponents()->front(), 0.0);
154     }
```

Figura3 - _execute (Hold.cpp)

```

103 void Hold::handleSignalReceived(int sigVal, int sigLimit) {
104     if (this->_type == Type::WaitForSignal) {
105         int expectedSig = _model->parseExpression(_waitForValue);
106         int limit = _model->parseExpression(_limit);
107         Waiting* first = _queue->first();
108
109         if (first != nullptr && expectedSig == sigVal) {
110             int releaseLimit;
111             if (sigLimit == 0)
112                 releaseLimit = limit;
113             else if (limit == 0)
114                 releaseLimit = sigLimit;
115             else
116                 releaseLimit = std::min(sigLimit, sigVal);
117
118             // ReleaseLimit == 0 means releaseAll at this point
119             if (releaseLimit == 0 || releaseLimit > _queue->size())
120                 releaseLimit = _queue->size();
121
122             while (releaseLimit > 0) {
123                 first = _queue->first();
124                 this->_queue->removeElement(first);
125                 this->releaseEntity(first->getEntity(), first->getTimeStartedWaiting());
126                 releaseLimit--;
127             }
128         }
129     }
130 }
131
132 /* Releases an entity from the Hold queue and sends it to the next component connected */
133 void Hold::releaseEntity(Entity* entity, double startedWaiting) {
134     double waitingTime = (_model->getSimulation()->getSimulatedTime() - startedWaiting);
135     entity->getEntityType()->getStatisticsCollector("Wait Time")->getStatistics()->getCollector()->addValue(waitingTime);
136     entity->setAttributeValue("Entity.WaitTime", entity->getAttributeValue("Entity.WaitTime") + waitingTime);
137     _model->sendEntityToComponent(entity, this->getNextComponents()->frontConnection(), 0.0);
138 }

```

Figura4 – handleSignalReceived e releaseEntity (Hold.cpp)

2.3 Signal.h

```

20 class Signal : public ModelComponent {
21 public:
22     Signal(Model* model);
23     Signal(const Signal& orig);
24     virtual ~Signal();
25
26     virtual std::string show();
27     static PluginInformation* GetPluginInformation();
28     static ModelComponent* LoadInstance(Model* model, std::map<std::string, std::string>* fields);
29
30     std::string getSignalValue() const;
31     std::string getLimitExpr() const;
32
33     void setSignalValue(std::string _signalValue);
34     void setLimitExpr(std::string _limit);
35
36 protected:
37     virtual void _execute(Entity* entity);
38     virtual bool _loadInstance(std::map<std::string, std::string>* fields);
39     virtual void _initBetweenReplications();
40     virtual std::map<std::string, std::string>* _saveInstance();
41     virtual bool _check(std::string* errorMessage);
42
43 private:
44     std::string _signalValue = "1.0";
45     std::string _limit = "0.0";
46
47 };

```

Figura5 – Signal.h

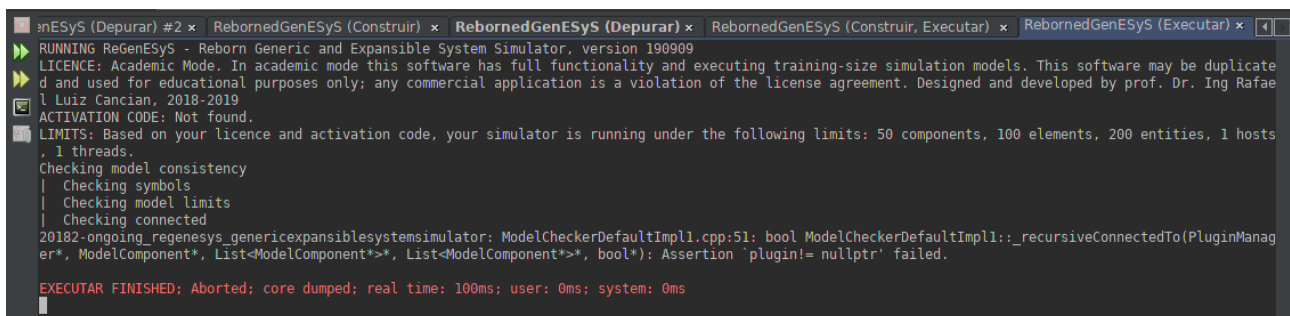
2.4 Signal.cpp

```
65 void Signal::_execute(Entity* entity) {
66     int limit = _model->parseExpression(_limit);
67     int sigVal = _model->parseExpression(_signalValue);
68
69     List<ModelElement*> *elements = _model->getElementManager()->getElements(Util::TypeOf<Hold>());
70     ModelElement* element = elements->front();
71
72     for (int i = 0; i < elements->size(); ++i) {
73         Hold* hold = dynamic_cast<Hold*>(element);
74
75         if (hold->getType() == Hold::Type::WaitForSignal) {
76             hold->handleSignalReceived(sigVal, limit);
77         }
78
79         element = elements->next();
80     }
81
82     _model->sendEntityToComponent(entity, this->getNextComponents()->frontConnection(), 0.0);
83 }
```

Figura6 - _execute (Signal.cpp)

3. Dificuldades

- Instanciação de um componente Hold num modelo qualquer;



```
RebornGenESyS (Depurar) #2 x | RebornGenESyS (Construir) x | RebornGenESyS (Depurar) x | RebornGenESyS (Construir, Executar) x | RebornGenESyS (Executar) x
>> RUNNING ReGenESyS - Reborn Generic and Expansible System Simulator, version 190909
>> LICENCE: Academic Mode. In academic mode this software has full functionality and executing training-size simulation models. This software may be duplicated and used for educational purposes only; any commercial application is a violation of the license agreement. Designed and developed by prof. Dr. Ing Rafael Luiz Cancian, 2018-2019
ACTIVATION CODE: Not found.
LIMITS: Based on your licence and activation code, your simulator is running under the following limits: 50 components, 100 elements, 200 entities, 1 hosts, 1 threads.
Checking model consistency
| Checking symbols
| Checking model limits
| Checking connected
20182-ongoing_regenesys_genericexpansiblesystemsimulator: ModelCheckerDefaultImpl1.cpp:51: bool ModelCheckerDefaultImpl1::_recursiveConnectedTo(PluginManager*, ModelComponent*, List<ModelComponent*>*, List<ModelComponent*>*, bool*): Assertion 'plugin!= nullptr' failed.
EXECUTAR FINISHED: Aborted; core dumped; real time: 100ms; user: 0ms; system: 0ms
```