SOBRAPO

PODes
Revista Eletrônica Pesquisa Operacional para o Desenvolvimento

Rio de Janeiro, v.6, n.2, p. 299-317, maio a agosto de 2014

DECISÃO COM REDES NEURAIS ARTIFICIAIS EM MODELOS DE SIMULAÇÃO A EVENTOS DISCRETOS

Marília Gonçalves Dutra da Silva[ab], Joao Jose de Assis Rangel[a*],

David Vasconcelos Corrêa da Silva[ab], Túlio Almeida Peixoto[a],

Ítalo de Oliveira Matias[a]

[a]*Universidade Candido Mendes – UCAM, Campos dos Goytacazes – RJ, Brasil*
[b]*Instituto Federal Fluminense – IFF, Campos dos Goytacazes – RJ, Brasil*

Resumo

O objetivo deste trabalho é avaliar frameworks para aplicação de redes neurais artificiais acoplados a modelos de simulação a eventos discretos com o software Ururau. A análise está centrada em encontrar mecanismos que possibilitem a realização de decisões através de outros algoritmos integrados ao código do modelo de simulação. Na literatura atual, há pouca citação de como aplicar algum tipo de inteligência computacional a partes do código de um modelo, que necessite refletir comportamentos de ações com aspectos de maior complexidade, como ações de pessoas em uma parte do processo. Por outro lado, muitos softwares de simulação permitem a customização dos modelos através de algoritmos que podem ser integrados aos modelos. Os resultados encontrados demonstraram que a utilização do framework ENCOG se mostrou adequado para a criação das redes neurais artificiais, sem apresentar erros de funcionamento e incompatibilidades com o código do software Ururau. Além disso, o trabalho permite que as pessoas interessadas em compreender esta estrutura possam visualizá-la diretamente no código do modelo proposto.

Palavras-chave: Ururau, Redes Neurais Artificiais, ENCOG, Decisão.

Abstract

The objective of this study is to evaluate frameworks for the application of artificial neural networks linked to models of discrete event simulation with the Ururau software. The research is focused on finding mechanisms to allow the implementation of decisions by other algorithms integrated into the code of the simulation model. In the current literature, there is little mention of how to apply some kind of computational intelligence to parts of the code of a model that needs to reflect behaviors of actions with aspects of higher complexity, as actions of people in a part of the process. On the other hand, many simulation softwares allow customization of the models through algorithms that can be integrated into the models. The results showed that the use of the ENCOG framework is adequate for the creation of artificial neural networks, without presenting operating errors and incompatibilities with the software code Ururau. Moreover, the study allows people interested in understanding this structure visualizing it directly in the code of the proposed model.

Key words: Ururau, Artificial Neural Networks, ENCOG, Decision.

***Autor para correspondência:** e-mail: joao@ucam-campos.br

Versão inicial submetida em 02/12/2013. Versão final recebida em 17/04/2014.

1. Introdução

Os softwares utilizados para construção de modelos de simulação a eventos discretos (SED), de uma forma geral, representam as decisões realizadas nos sistemas sob análise com regras básicas de operadores lógicos. Estes operadores realizam funções, como: "<" (menor que), ">" (maior que), "==" (igual a), "E", "OU" e suas combinações. Ou ainda, em outros casos mais simples, utilizam apenas porcentagens obtidas em dados históricos para representar as tendências realizadas pelo conjunto das decisões tomadas. De certa forma, alguns aspectos dos sistemas que configuram algumas decisões mais sofisticadas, tomadas por pessoas, por exemplo, são representadas de forma simplificada nestes modelos. Assim, a possibilidade de representar tais ações de maneira mais realística e "inteligente" é um desafio que se apresenta e tem motivado pesquisadores como Robinson *et. al* (2001) e Bergmann *et al.* (2014).

Segundo Swain (2007), quase 71% dos softwares utilizados para a construção de modelos de SED permitem a customização dos modelos através de algoritmos provenientes de outras linguagens ou módulos externos. No trabalho de Bergmann *et al.* (2014), foram utilizadas redes neurais artificiais (RNA) em conjunto com modelos de simulação. Neste caso, a geração dos modelos pode ocorrer de forma automática, e se baseia na aquisição de dados armazenados em sistemas de ERP (*Enterprise Resource Planning*). Assim, uma dificuldade encontrada nesta tarefa é a representação do comportamento dinâmico nos sistemas de manufatura, por exemplo. Diversas informações sobre comportamentos, como estratégias de controle, não são comumente armazenadas pelos sistemas de TI. Neste caso, é preciso a utilização de algoritmos específicos para extrair comportamentos complexos, e, muitas vezes, é necessário um especialista no sistema para adicionar o comportamento detalhado ao modelo que foi gerado automaticamente.

Segundo Zuben (2003), o comportamento humano pode ser emulado por uma RNA a partir de amostras de entradas e saídas de um sistema. Assim, o vetor de entrada contém o conjunto de informações recebidas por uma pessoa, por exemplo, e o vetor de saída, as respectivas ações tomadas por ele com base nas informações recebidas. Em outro trabalho sobre o assunto, Silva *et al.* (2012) utilizaram RNA para representar o comportamento de decisões mais sofisticadas, ou seja, com maior nível de detalhes, normalmente tomadas por pessoas em modelos de SED. Os autores realizaram os testes utilizando o software Ururau (Peixoto *et al.*, 2013). Neste caso, foi criado um módulo com um algoritmo de uma RNA que se comunicava através de soquetes TCP (*Transmission Control Protocol)*, com o modelo de simulação, capaz de representar de forma mais realística as decisões realizadas por pessoas nos sistemas modelados.

Diante do que foi apresentado, o objetivo deste trabalho foi buscar um mecanismo para incorporar ao software Ururau um módulo capaz de executar uma RNA diretamente no código de um modelo de simulação. A utilização deste módulo pode facilitar a elaboração de modelos de simulação e evitar a comunicação dos mesmos com algoritmos executados externamente. Desta forma, podem-se representar partes de processos modelados, onde a decisão é tomada com base em fatores diferentes dos operadores lógicos ou porcentagens, que tenham a ver com conhecimentos e experiências prévias do decisor.

Para isso, foram pesquisados e avaliados *frameworks* para aplicação de RNA em modelos de SED. Isso permitiu a realização de um estudo mais aprofundado sobre recursos e tecnologias já consolidadas. A princípio, o que se buscou, ao realizar a pesquisa, foi encontrar algum *framework* em Java puro que permitisse a criação, treinamento e execução de uma RNA. A necessidade de ser um *framework* Java se deu pelo fato do software Ururau ter sido desenvolvido originalmente nesta linguagem.

2. Referencial Teórico

2.1. O Ururau

O Ururau é um software de simulação de código aberto e livre de custos, que utiliza como base a biblioteca de simulação JSL (*Java Simulation Library)* (Rosseti, 2008). O software permite a construção de modelos de simulação tanto em interface gráfica (GUI - *Graphic User Interface*) quanto em uma API - *Application Programming Interface* (Peixoto *et al.*, 2013).

A Figura 1 apresenta a arquitetura do Ururau. Observe que a linguagem Java permeia todas as camadas que o compõe. Já, a biblioteca JSL, que está na camada mais inferior, converte o modelo para uma sequência de eventos discretos.

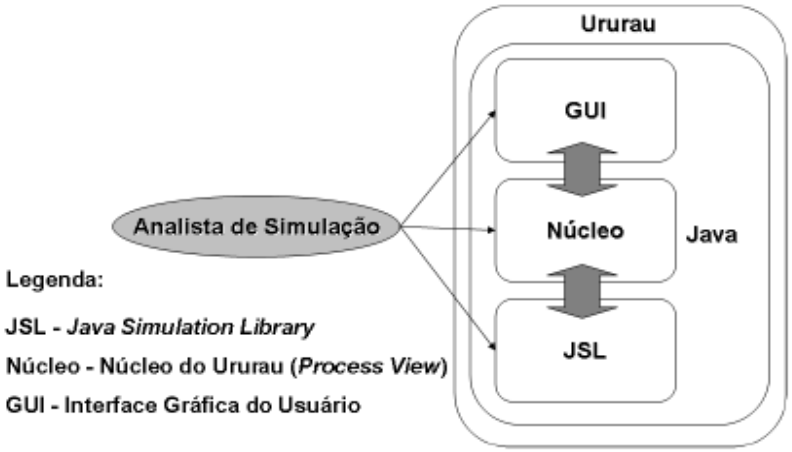

Figura 1: Arquitetura do Ururau. (Fonte: Peixoto, *et al.*(2013)).

O núcleo do Ururau está na camada intermediária, sendo composto por comandos de processos específicos do JSL. A camada mais superior trata da conversão do modelo gráfico, que é composto por um grafo dirigido para uma sequência de comandos do núcleo do software. Observe que o analista de simulação pode atuar em qualquer camada do software de acordo com a necessidade observada no momento de elaboração do modelo.

Silva *et al.* (2012) realizaram uma extensão ao Ururau partindo de suas camadas mais baixas (JSL e Núcleo) de forma a possibilitar a comunicação deste com o módulo inteligente. Para testar este mecanismo, os autores desenvolveram uma comunicação entre o Ururau e o módulo inteligente baseada em soquetes TCP, conforme ilustra a Figura 2.

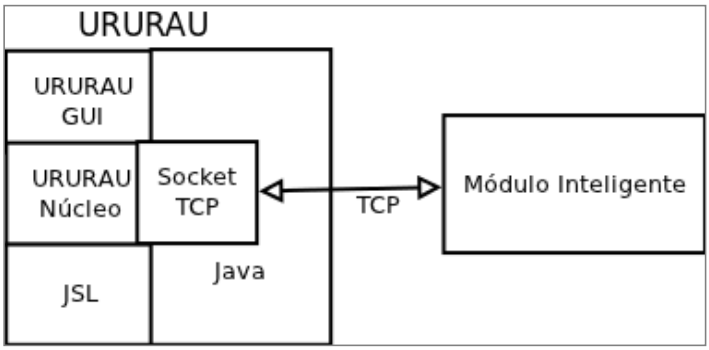

Figura 2: Forma de comunicação do módulo inteligente com o modelo. (Fonte: Silva *et al.* (2012)).

Na abordagem proposta por Silva *et al.* (2012), o modelo de simulação foi criado no Ururau a partir de linhas de código, e este, quando executado, enviava os dados relacionados ao processo decisório para o módulo inteligente. Ao receber os dados, o módulo inteligente executava uma RNA, previamente treinada, para retornar a decisão a ser executada pelo modelo de simulação.

2.2. Frameworks de Redes Neurais Artificiais em Java

De acordo com Baptista & Dias (2012), a escolha da solução mais conveniente para a aplicação envolvendo RNA pode ser uma tarefa complexa. A avaliação deve considerar a arquitetura da RNA, o algoritmo de treinamento, o sistema operacional, e o tipo de licença do software. Desta forma, três dos principais *frameworks* não comerciais foram pesquisados. São eles: JOONE, NEUROPH, e ENCOG.

2.2.1 JOONE

O JOONE (*Java Object Oriented Neural Engine*) é um *framework* escrito em linguagem Java para a construção e execução de aplicações de Inteligência Artificial baseadas em RNA. Pode ser executado em qualquer plataforma e é compatível com vários sistemas operacionais como: Linux, Mac OSX, Windows 2000, Windows XP e SUN Solaris (MARRONE, 2007).

O software é composto por um motor central, um editor de GUI e um ambiente de treinamento distribuído (JOONE, 2013). De acordo com Marrone (2007), esta ferramenta está licenciada sob a LGPL (GNU *Lesser General Public License*), desenvolvida pela comunidade JOONE, e pode ser utilizada tanto por entusiastas quanto por usuários profissionais.

Marrone (2007) afirma ainda que o JOONE possibilita que as RNAs sejam criadas em uma máquina local, treinadas em ambiente distribuído e executadas em qualquer dispositivo. O mesmo é desenvolvido em componentes que possuem algumas características básicas, tais como, persistência, *multithreading*, serialização e parametrização, o que garante a escalabilidade, confiabilidade e expansibilidade.

2.2.2 NEUROPH

O NEUROPH é um *framework* para desenvolvimento de RNA que consiste em uma biblioteca Java e um editor de RNA denominado NEUROPH *Studio* (Sevarac & Koprivica, 2013). É um projeto *open source* hospedado no repositório *SourceForge* e, desde a versão 2.4, é licenciado sob Apache 2.0. Versões anteriores estão sob a licença LGPL3 (SEVARAC & KOPRIVICA, 2013).

2.2.3 ENCOG

O ENCOG é um *framework* de Inteligência Artificial Java, .Net e C/C++. Inicialmente, o ENCOG foi criado para suportar somente RNA, contudo, com o tempo, foi se expandindo para trabalhar com aprendizagem de máquina em geral (HEATON, 2011).

De acordo com Heaton (2011), o ENCOG faz uso de diversas bibliotecas de terceiros para adicionar funcionalidades necessárias, sem "reinventar a roda".

O ENCOG possui uma aplicação gráfica conhecida como ENCOG *Workbench*, que permite realizar diversas tarefas de aprendizagem de máquina, sem que seja necessário escrever código Java ou C#. Ele é escrito em Java, mas gera arquivos que podem ser usados com qualquer *framework* ENCOG.

2.2.4 Comparação dos Frameworks de RNA Java

Baptista & Dias (2012) afirmam que as informações contidas em artigos e sites que tratam das ferramentas de RNA são suficientes para que se possa tomar uma decisão a respeito daquela que melhor se encaixa ao problema a ser resolvido.

Matviykiv & Faitas (2013) realizaram uma análise com o objetivo de escolher a melhor ferramenta para o desenvolvimento de uma RNA para classificação espectral. Os autores testaram o ENCOG 3.1, o JOONE 2 RC1, o NEUROPH 2.6, e também a ferramenta FANN 2.2, que não é objeto deste estudo. Assim, com base na análise apresentada, os autores puderam concluir que o ENCOG apresentou melhores resultados e maior facilidade de utilização.

Baptista & Dias (2013) apresentaram dados sobre um grande número de ferramentas que podem ser utilizadas para criação de RNA. O trabalho desenvolvido pelos autores levou em consideração os seguintes aspectos: sistema operacional, requisitos mínimos de hardware e software, licença, arquitetura da rede e algoritmo de treinamento.

O Quadro 1 apresenta a relação dos sistemas operacionais suportados por cada *framework* em estudo.

Nota-se que todas as ferramentas são compatíveis mais de um sistema operacional, o que, na prática, garante uma maior interoperabilidade da ferramenta. Em relação aos requisitos mínimos de software e *hardware* necessários para o funcionamento das ferramentas, Baptista & Dias (2013) apresentam poucas informações a respeito dos *frameworks* em estudo.

Quadro 1: Sistemas Operacionais (Fonte: Adaptado de Baptista & Dias (2013)).

Frameworks	**Sistemas Operacionais**					
	Windows	**Mac OSX**	**Unix**	**Linux**	**Sun**	**Outros**
ENCOG	sim	sim	sim	sim	não	não
JOONE	sim	sim	não	sim	sim	não
NEUROPH	sim	não	não	sim	não	não

No que diz respeito ao JRE (*Java Runtime Environment)*, o ENCOG precisa ter, no mínimo, versão 1.5 instalada. O NEUROPH funciona somente com versão a partir de 1.6. Quanto ao JOONE, não é apresentada nenhuma informação sobre o JRE, porém, é mencionada a necessidade de, no mínimo 256 MB de memória RAM (*Random Access Memory).* Este fato deve estar associado à escassez de informações sobre o assunto nas diversas fontes consultadas pelos autores.

O Quadro 2 traz a relação das arquiteturas de redes que podem ser desenvolvidas por cada *framework*.

Quadro 2: Arquitetura de Rede (Fonte: Adaptado de Baptista & Dias (2013)).

Frameworks	**Arquitetura de Rede**
ENCOG	*Adaline Linear Neuron; Adaptive Resonance Theory; Bidirectional Associative Memory; Boltzmann Machine; Counter-Propagation Neural Network; Elman Network; Hopfield Network; Jordan Recurrent; Kohonen Networks; Multilayer Perceptron; Neuro evolution of Augmenting Topologies; Radial Basis Function; Self-Organizing Map.*
JOONE	*Feed-Forward Neural Network; Kohonen Networks; Modular Neural Network; Principal Component Analysis; Time-Delay Neural Network*
NEUROPH	*Adaline Linear Neuron; Bidirectional Associative Memory; Competitive Neural Network; Hebbian Network; Hopfield Network; Kohonen Networks; Maxnet; Multilayer Perceptron; Radial Basis Function.*

Baptista & Dias (2013) afirmam que a arquitetura de uma RNA está relacionada com a estrutura e o tipo da rede, e que é uma das decisões mais importantes no momento da escolha da ferramenta. Algumas redes são mais apropriadas para alguns tipos de tarefas do que outros, e, desta forma, cada ferramenta abrange apenas um subconjunto de todas as arquiteturas disponíveis.

Outro aspecto importante e destacado pelos autores são os algoritmos de treinamento. O Quadro 3 apresenta os resultados encontrados pelos autores.

Quadro 3: Algoritmo de Treinamento (Fonte: Adaptado de Baptista & Dias (2013).)

Frameworks	**Algoritmos de Treinamento**
ENCOG	*Backpropagation; Conjugate Gradient; Competitive Learning; Genetic Algorithm; Levenberg–Marquardt.*
JOONE	*Backpropagation; Batch Training; Conjugate Gradient Descent; Delta-bar-Delta; Resilient Propagation.*
NEUROPH	*Backpropagation; Competitive Learning.*

O ENCOG e o JOONE apresentaram a mesma quantidade de algoritmos de treinamento, embora com diferenças entre ambos. Somente o algoritmo *backpropagation* pode ser utilizado por todos os *frameworks*. Embora não esteja listado no trabalho de Baptista & Dias (2013), o algoritmo de treinamento *Resilient Propagation* também está disponível no ENCOG 3, de acordo com informações apresentadas por Heaton (2010).

O Codeproject (2010a) realizou uma comparação entre o NEUROPH, o ENCOG e o JOONE. Para tal, foi criada uma RNA *feedforward* para reconhecer uma operação XOR.

As versões testadas e comparadas de cada f*ramework* são as seguintes: ENCOG v2.4, NEUROPH v2.4 e JOONE v2 RC1. O computador utilizado foi um *Dell Studio* XPS 8000, com processador *Intel Core* i7 860@ 2.8ghtz. O processador é *quadcore* com *hyperthreading*.

Segundo o autor, a ideia da realização dos testes se baseou na observação de que o *framework* JOONE, com o qual o mesmo trabalhava, apresenta uma alta complexidade no código fonte. O objetivo foi verificar se outro *framework* seria capaz de realizar ciclos de treinamento de forma mais rápida fazendo uso do recurso *multicore* das máquinas modernas. O Quadro 4 apresenta algumas informações importantes sobre a RNA criada para os testes:

Quadro 4: Dados de criação da RNA (Fonte: Codeproject (2010 a)).

Dados da RNA	
Neurônios de Entrada	10
Neurônios de Saída	10
Neurônios da Camada Oculta	20
Função	TANH
Conjunto de Treinamento	100 elementos
Iteração	50
Método de Treinamento	*backpropagation* com momento

A Figura 3 apresenta a comparação do tempo total de processamento das RNA criadas utilizando cada um dos *frameworks*. O ENCOG e o JOONE oferecem recursos *multithread*, que possibilitam o processamento paralelo beneficiado, por exemplo, pela tecnologia *multicore* presente nos computadores atuais. O NEUROPH não apresenta essa funcionalidade. O recurso *multithread* está representado pelas letras MT (*multithread*) e ST (*monothread)* quando não tiver o recurso.

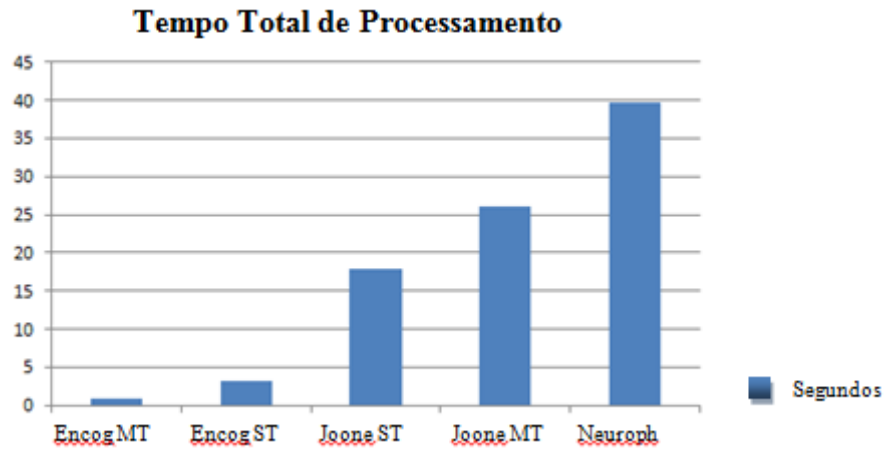

Figura 3: Tempo total de Processamento da RNA (Fonte: Adaptado de Codeproject (2010 a)).

De acordo com o resultado dos testes realizados por Codeproject (2010 a) e apresentados na Figura 3, é possível notar que o ENCOG teve os menores tempos de processamento, tanto em modo MT quanto em modo ST.

O JOONE mostrou um desempenho inferior trabalhando em modo MT, o que significa que, com uma arquitetura desenvolvida utilizando múltiplas *threads*, o processamento da RNA pode levar mais tempo, mesmo sendo utilizada uma máquina *multicore*.

A RNA criada utilizando o NEUROPH foi a que gastou maior tempo para ser processada.

A partir dos testes realizados, o Codeproject (2010a-b) apresentou algumas conclusões, cujas mais relevantes estão relatadas aqui. No geral, o ENCOG e o NEUROPH apresentaram melhores resultados que o JOONE, porém, o *framework* que apresentou superioridade na maioria dos testes foi o ENCOG.

Foram necessárias apenas 18 iterações para o treinamento da rede utilizando o ENCOG, enquanto o NEUROPH realizou 613 iterações e o JOONE mais de 5000 iterações. Talvez essa diferença na quantidade de iterações do ENCOG, em relação aos outros dois *frameworks*, esteja relacionada com o fato do ENCOG utilizar GPU (*Graphics Processing Unit*, ou Unidade de Processamento Gráfico) para aumentar sua velocidade de treinamento.

2.3. Conclusões do Referencial Teórico

As conclusões obtidas com os trabalhos de Baptista & Dias (2013), Matviykiv & Faita (2013) e Codeproject (2010 a-b) possibilitaram observar que o ENCOG apresentou melhores resultados em relação à execução da RNA e, também, na maioria dos quesitos da comparação.

Durante as buscas e pesquisas realizadas, foi possível observar que o ENCOG possui uma grande gama de informações disponibilizadas sob forma de tutoriais, fóruns, listas de discussões e livros publicados, o que, em termos software livre, garante uma maior perspectiva de continuidade, crescimento e desenvolvimento de novos recursos e melhorias da ferramenta.

Por meio deste estudo de pesquisa envolvendo os três *frameworks*, foi possível observar que o ENCOG se apresentou como o *framework* mais completo em termos de funcionalidades, e também o que apresentou melhor desempenho.

3. Metodologia

Ao buscar o *framework* mais adequado ao projeto, os seguintes fatores foram considerados como tendo grande importância:

a) Interoperabilidade – capacidade de funcionar em vários sistemas operacionais;
b) *Framework* Java – compatibilidade com o código do Ururau existente;
c) Possuir código fonte aberto – possibilidade de expansão e melhorias; uso acadêmico;
d) Ser gratuito – uso acadêmico;
e) Possuir facilidade de acesso a informações, atualizações, comunidade engajada – indicativo de melhorias, atualizações, expansão; crescimento; uso acadêmico.

A proposta deste trabalho está centrada em avaliar a possibilidade de acoplamento do módulo contendo um algoritmo de uma RNA a toda estrutura interna do código de um modelo de simulação em Ururau. A Figura 4 ilustra esta proposta. Note que a ideia é que o próprio modelador possa manipular o módulo inteligente através da camada mais superior do software, ou seja, através de módulos da GUI.

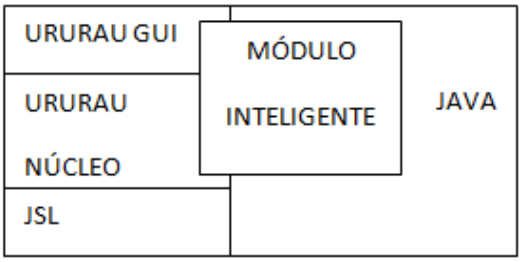

Figura 4: Módulo Inteligente acoplado ao Ururau.

A incorporação do módulo inteligente ao Ururau possibilita, entre outras coisas, que a RNA tenha seus parâmetros configurados no momento da criação do modelo, e pela GUI.

Observe, ainda, que o usuário também tem a opção de poder manipular a RNA nas camadas do núcleo, por meio de linhas de código em Java. Com base nos critérios preestabelecidos, e nas pesquisas realizadas, o ENCOG foi o *framework* escolhido para o desenvolvimento do módulo inteligente no Ururau.

A etapa seguinte à escolha do *framework* foi a realização de testes para saber se a ferramenta seria realmente adequada. Para tal, foram realizados testes na ferramenta, e, inicialmente, duas principais perguntas foram feitas:

a) O ENCOG funciona sem falhas ou inconsistências junto ao Ururau?
b) O módulo implementado com o ENCOG está gerando os resultados corretos?

Para responder a primeira pergunta, foi realizado um teste, chamado teste de acoplamento. O objetivo deste teste foi identificar possíveis falhas e conflitos entre o código existente do Ururau e o código do *framework* ENCOG. Nesta etapa, também foi possível fazer

pequenos ajustes nos códigos, e verificar se o Ururau continuaria "rodando" sem a ocorrência de problemas de incompatibilidades. A princípio, o objetivo foi que os resultados obtidos no teste fossem analisados a partir de um modelo bem simples.

A etapa seguinte foi chamada de teste de funcionamento, que tem relação com a segunda pergunta. Neste caso, não bastaria que os códigos do Ururau e do ENCOG, já unificados, não gerassem conflitos, mas que os resultados obtidos durante a execução de modelos de simulação utilizando RNA gerassem os resultados corretos. Para isso, foi importante partir de algum modelo já validado e realizar comparações em relação aos resultados obtidos.

Os próximos subitens detalham os passos dos testes descritos.

3.1. Teste de Acoplamento

O primeiro teste realizado com o ENCOG teve como objetivo verificar se o mesmo possui um bom acoplamento com o Ururau em termos de compatibilidade entre os códigos.

Para tal, o ENCOG foi incorporado ao núcleo do Ururau. Pequenas alterações e ajustes foram realizados de forma a interligar o núcleo existente do Ururau e o código do ENCOG. Passada esta primeira etapa de ajuste dos códigos, foi realizado um teste a partir de um modelo hipotético que representa uma decisão "ou exclusivo", (XOR). Neste primeiro teste, foi criada uma RNA XOR conforme modelo conceitual apresentado na Figura 5.

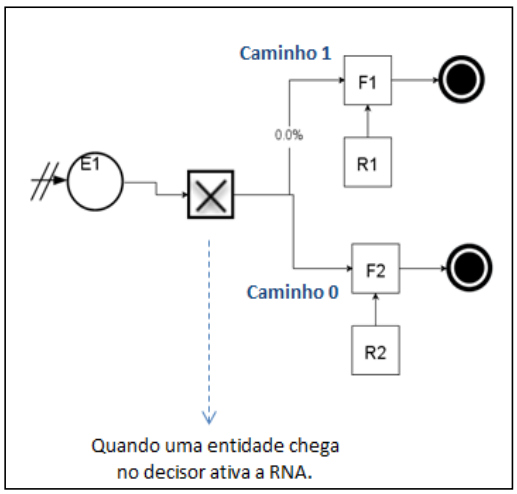

Figura 5: Modelo Conceitual de uma rede XOR no Ururau.

O referido modelo está representado em linguagem IDEF-SIM (Montevechi *et al.*, 2010). É importante citar que o Ururau utiliza componentes gráficos semelhantes aos elementos

desta linguagem. Essa característica torna a conversão do modelo conceitual em modelo de simulação de entendimento mais fácil e direto.

No problema modelado, quando uma entidade encontra o operador de decisão, a RNA é ativada e a mesma consulta os valores dos neurônios de entrada. A tabela verdade da função XOR funciona como demonstrado na tabela 1:

Tabela 1: Tabela Verdade da Função XOR.

Neurônios de Entrada		**Neurônio de Saída**
N1	**N2**	**S**
0	0	0
0	1	1
1	0	1
1	1	0

A arquitetura da RNA criada neste teste é apresentada na Figura 6, em que os neurônios de entrada obtêm dados resultantes de expressões processadas pelo Ururau, como por exemplo: tamanho da fila, tempo na fila, testes condicionais, entre outras.

No exemplo representado pela Figura 5, F1 representa a quantidade de entidades na fila do processo F1, e F2 representa a quantidade de entidades na fila do processo F2. O caminho a ser percorrido no modelo pela entidade é definido segundo a decisão da própria RNA e não mais por critérios pré-definidos. Após a execução da RNA, cada entidade segue seu percurso no modelo de acordo com a decisão tomada pela mesma.

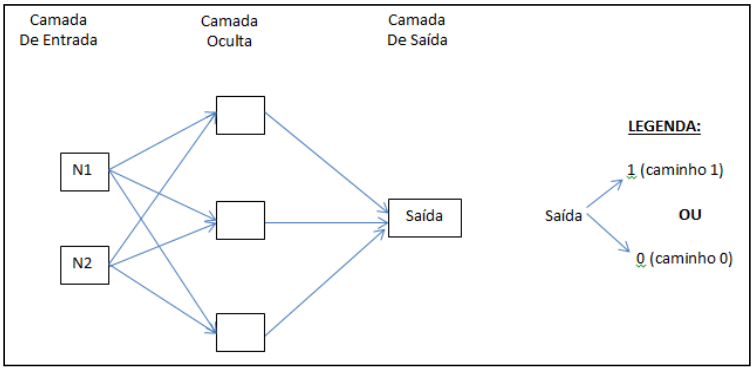

Figura 6: Exemplo da RNA XOR implementada no teste.

Os neurônios da camada de entrada foram configurados de forma que receberão o valor binário 1, no caso de existirem entidades aguardando na fila dos processos F1 e F2, e receberão valor binário 0, no caso de não haver entidades aguardando na fila.

Por se tratar de um problema binário, convencionou-se que, no caso do valor de saída da RNA ser 0, a entidade é encaminhada para o caminho 0 do modelo, e, no caso da saída da RNA ser o valor 1, a entidade será encaminhada para o caminho 1 do modelo (Figura 6).

Neste caso, supondo-se, por exemplo, que, em um dado momento da simulação, F1 seja igual a 1, ou seja, há uma entidade aguardando na fila F1, e F2 seja igual a 0, ou seja, não há entidades aguardando na fila F2, temos que:

N1= "F1>0", resulta 1 na entrada do primeiro neurônio (N1);

N2= "F2>0", resulta 0 na entrada do segundo neurônio (N2).

Desta forma, temos as seguintes entradas nos neurônios: N1 = 1 e N2 = 0. Como a RNA implementa um XOR, temos a SAÍDA = 1. Na simulação, a entidade deve seguir pelo CAMINHO 1.

3.2. Teste de Funcionamento

Objetivando verificar o funcionamento do novo módulo inteligente (acoplado ao Ururau), decidiu-se criar um modelo de simulação já validado por Silva *et al.* (2012), de forma que fosse possível afirmar se o módulo inteligente acoplado ao Ururau estaria gerando os resultados corretos.

Para isso, o mesmo modelo de simulação foi gerado tanto no módulo acoplado quanto no módulo inteligente desenvolvido por Silva *et al.* (2012), de modo a permitir a comparação de resultados. É importante destacar que se optou por utilizar uma massa de dados diferente da utilizada por Silva *et al.* (2012).

A Figura 7 apresenta as etapas da simulação como módulo inteligente se comunicando por meio de soquete TCP e as respectivas ferramentas necessárias para a realização das mesmas.

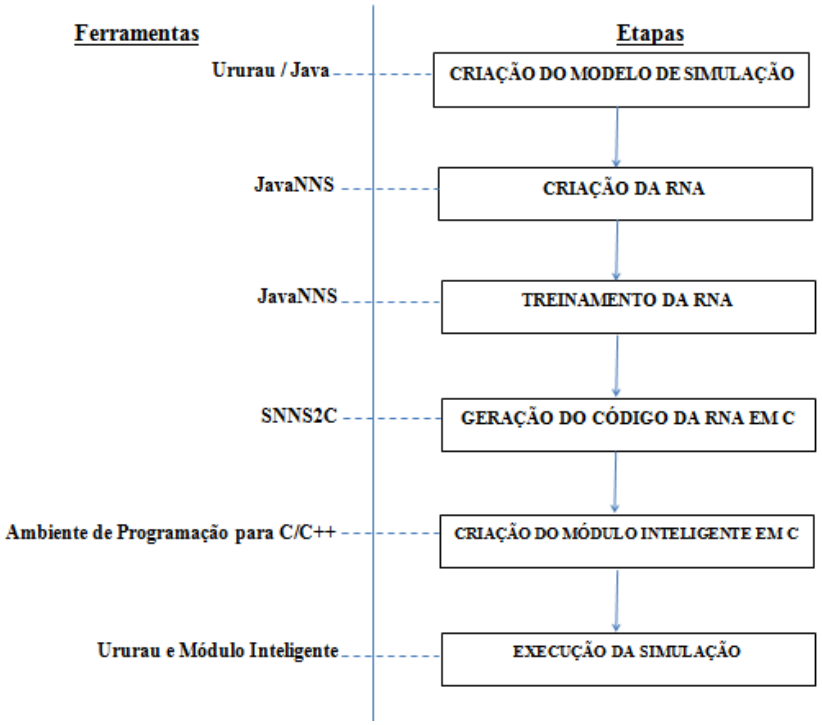

Figura 7: Ferramentas e Etapas da Simulação Inteligente.

Conforme pode ser observado na Figura 7, a primeira etapa a ser realizada é a criação do modelo de simulação, utilizando-se o Ururau. O modelo de simulação que foi utilizado para simulação e comparação dos resultados obtidos em ambos os módulos inteligentes foi o modelo desenvolvido por Silva *et. al* (2012) em seu trabalho, conforme pode ser visualizado conceitualmente na figura 8.

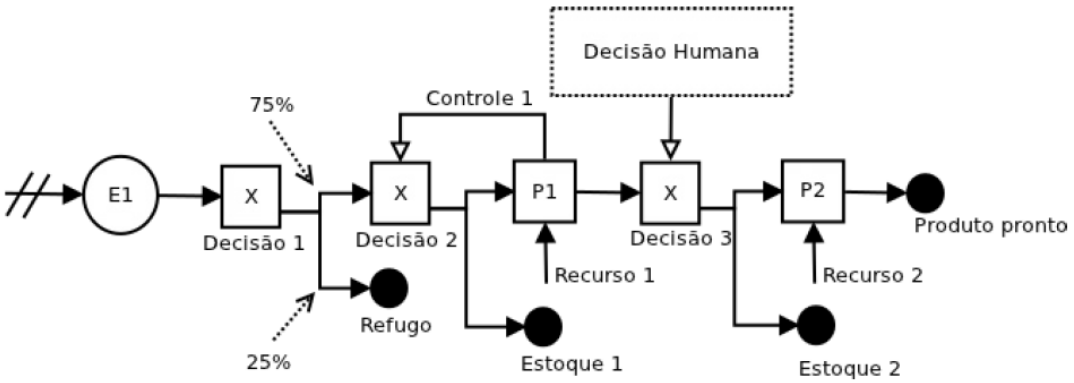

Figura 8: Modelo Conceitual em IDEF-SIM com "decisão 3" usando RNA (Fonte: Silva, *et al.* (2012)).

Após a criação do modelo de simulação, foi necessária a criação da RNA, que, neste caso, foi realizada utilizando o JavaNNS (*Java Neural Network Simulator*). Para a construção

desta RNA, Silva *et al*. (2012) utilizaram três neurônios na primeira camada, sendo um neurônio para cada parâmetro.

Definiu-se que o "operador" tomaria suas decisões com base nos seguintes parâmetros: tamanho da fila P2; tempo de vida da entidade E1; e turno (Figura 9).

Além disso, foram criados três neurônios na camada oculta e um neurônio na camada de saída. Após a RNA criada, foi necessário que a mesma fosse treinada. O treinamento também foi realizado utilizando-se a ferramenta JavaNNS.

No trabalho proposto por Silva *et al.* (2012), foi necessário que, após o treinamento da RNA, fosse gerado um código da mesma em linguagem de programação C. Depois da geração do código em C da RNA, este código foi incorporado ao módulo inteligente, que, por sua vez, pôde interagir com o código do modelo em Ururau, através de uma comunicação com soquetes TCP.

Quando o modelo foi executado, os dados utilizados para a tomada de decisão foram enviados para o módulo inteligente. A RNA foi responsável por definir a decisão, que, por sua vez, é enviada de volta para o modelo em Ururau, fechando, desta forma, o ciclo de operação.

Por outro lado, a partir da utilização do *framework* ENCOG e do acoplamento realizado, foi possível criar o código Java da decisão compatível com o modelo em Ururau. Esse código foi executado todo em memória interna do computador. Assim, as etapas da simulação puderam ser reduzidas em apenas duas. A Figura 9 apresenta a nova estrutura desenvolvida, em que todas as etapas ocorrem diretamente no software Ururau.

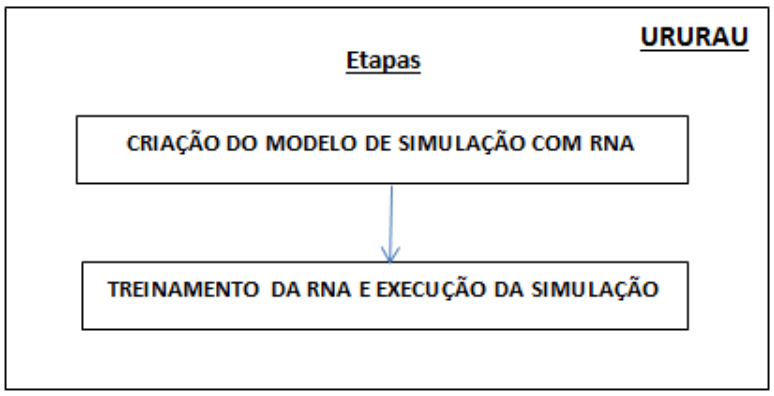

Figura 9: Etapas da Simulação Inteligente

A mesma massa de dados foi submetida a testes no novo módulo inteligente. O modelo utilizado também foi o mesmo de Silva, *et al.*, apresentado na Figura 8. Os resultados obtidos nos testes realizados são discutidos na seção seguinte.

4. Resultados e Discussão

Com a realização do primeiro teste, foi possível observar que o ENCOG funcionou sem problemas de incompatibilidades com o código do Ururau. A RNA criada também apresentou os resultados esperados para os cenários possíveis testados.

Partindo do princípio que os ajustes realizados nos códigos do Ururau e do ENCOG foram realizados com sucesso no processo de acoplamento, e que os resultados obtidos no primeiro teste estavam de acordo com o esperado, deu-se prosseguimento para a etapa seguinte. Desta forma, ao executar o modelo em ambos os módulos inteligentes, os resultados obtidos no teste podem ser visualizados conforme Quadro 5, que segue.

Comparando os resultados da simulação da rede neural JavaNNS (Silva, *et al.* 2012) com o módulo inteligente desenvolvido com o ENCOG, observa-se que, das 2116 decisões, ambas RNA tomaram a mesma decisão em 2106 das vezes.

Quadro 5: Resultados dos Experimentos 1 e 2.

Parâmetro	**Experimento 1**	**Experimento 2**
Ferramenta de RNA	**JavaNNS**	**ENCOG**
Tempo médio das entidades na fila P2	233,67 min	235,22 min
Quantidade média de entidades na fila P2	12,01 un	12,20 un
Recurso R2 ocupado	96,48%	96,86%
Decisões com RNA	2116 unidades	
Decisões divergentes entre o experimento 1 e 2	10 unidades	
Percentual de decisões divergentes	0,47%	

Em relação ao modelo de simulação, observou-se uma pequena variação entre os valores de “Tempo médio das entidades na fila P2”; “Quantidade média de entidades na fila P2”; e “Recurso R2 ocupado” obtidos nos dois experimentos. Contudo, é difícil afirmar se tais variações são consequência das decisões divergentes entre as RNA, que ocorrem em apenas 0,47% das vezes, ou da aleatoriedade dos sistemas estocásticos.

5. Conclusões

Os resultados deste trabalho mostraram que o *framework* ENCOG permitiu a realização do acoplamento de uma rede neural artificial com um modelo de simulação a eventos discretos executado no software Ururau. Ou seja, o respectivo *framework* se mostrou adequado para a criação de um módulo inteligente, sem apresentar erros de funcionamento e incompatibilidades com o código do modelo de simulação. Em relação ao teste comparativo realizado com o módulo proposto e o apresentado pela literatura, foi possível observar que os valores dos resultados encontrados foram compatíveis, mostrando que o ENCOG foi capaz de executar a rede neural de maneira confiável acoplado ao software de simulação utilizado.

O principal benefício obtido com os resultados deste trabalho foi possibilitar que um desenvolvedor construa um modelo de simulação a eventos discretos que tenha algoritmos com decisões inteligentes inseridos no próprio modelo. Ou seja, permitir que um modelador construa um modelo de simulação com uma decisão inteligente baseada em rede neural utilizando somente o software Ururau. Como mostrado em trabalhos anteriores, para se alcançar resultado semelhante era necessário que o desenvolvedor utilizasse ferramentas como: Ururau, JavaNNS, SNNS2C, e ambiente de programação C/C++.

Em relação a trabalhos futuros, novos ajustes e adaptações estão sendo realizados do Ururau, de forma a permitir que modeladores menos familiarizados com a linguagem Java, possam elaborar modelos de simulação com decisões com redes neurais, apenas pela interface gráfica do software. Além disso, o ENCOG permite que outras funcionalidades baseadas em inteligência computacional sejam adicionadas. Assim, por se tratar de um projeto com software de código aberto e livre de custos, os autores convidam outros pesquisadores a cooperarem nesta questão.

Agradecimentos

Os autores gostariam de agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e à Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) pelo suporte financeiro para esta pesquisa.

Referências

Baptista, D. & Dias, F.M. (2013). A survey of artificial neural network training tools. Neural Comput & Appic. New Applications of Artificial Neural Network in Modeling & Control. Springer-Verlag, London.

Baptista, D. & Dias, F.M. (2012). Artificial Neural Networks: A Review of Training Tools . 10Th Portuguese Conference on Automatic Control, Portugal.

Bergmann, S. & Stelzer, S. & Strassburger, S. (2014). On the use of artificial neural networks in simulation-based manufacturing control. Journal of Simulation, 8, 76-90.

Codeproject. Benchmarking and Comparing ENCOG, NEUROPH and JOONE Neural Networks. (2010 - a). Disponível em: http://www.codeproject.com/Articles/85487/

Benchmarking-and-Comparing-Encog-NEUROPH-and-JOONE. Acesso em: 28/06/2013.

Codeproject. Comparing Neural Networks in NEUROPH, ENCOG and JOONE. (2010 - b). Disponível em: http://www.codeproject.com/Articles/85385/Comparing-Neural-Networks-in-Neuroph-Encog-and-JOO. Acesso em: 28/06/2013.

Heaton, J. (2010). Introduction to ENCOG 2.5 . Heaton Research, Inc. 108p. Disponível em: Http://Www.Heatonresearch.Com/Encog. Acesso: 15 de Julho de 2013.

Heaton, J. (2011). Programming Neural Networks with Encog3 in Java. Heaton Research, Inc. St Louis, MO, USA. 242p.

JOONE. (2013). An Object Oriented Neural Engine. Disponível em: http://sourceforge.net/projects/joone/. Acesso em 03 de Junho de 2013.

Marrone, P. (2007). JOONE Java Object Neural Engine. The Complete Guide: All you need to know about JOONE. 142 p. Disponível em: http://www.joone.org.

Matviykiv,O.M & Faitas, O.I., (2013). Data Classification of Spectrum Analysis Using Neural Network. Lviv Polytechnic National University, Computer-Aided Design Department.

Montevechi, J.A.B. & Leal, F. & De Pinho, A.F. & Da Silva, R.F.C. & De Oliveira, M.L.M. & Da Silva, A.L.F. (2010). Conceptual modeling in simulation projects by mean adapted IDEF: An application in a Brazilian tech company. In: Winter Simulation Conference, p. 1624-1635.

Peixoto, T.A. & Rangel, J.J.A. & Matias, I. O. & Montevechi, J.A.B. & Miranda, R.C. (2013). Ururau – Um Ambiente para Desenvolvimento de Modelos de Simulação a Eventos Discretos. PODes - Revista Eletrônica Pesquisa Operacional para o Desenvolvimento. Vol.5, n.3, p.373-405.

Sevarac, Z & Koprivica, M. (2013). Getting Starded With NEUROPH. Retirado de: http://neuroph.sourceforge.net. Acesso em 13/07/2013.

Silva, D.V.C. & Rangel, J.J.A. & Matias, I.O. & Vianna, D.S. & Peixoto, T.A. (2012). Modelos de Simulação a Eventos Discretos com Aspectos de Decisão Humana: Uma Aplicação com o Ururau. PODes - Revista Eletrônica Pesquisa Operacional para o Desenvolvimento. Vol.4, n.3, p.339-355.

Robinson, S. & Alifantis, T. & Hurrion, R. & Edwards, J. S. & Ladbrook, J. & Waller, T. (2001). Modelling and Improving Human Decision Making With Simulation. In: Winter Simulation Conference, IEEE, Arlington, P. 913-920.

Rosseti, M. D. (2008). Java Simulation Library (JSL): An Open-Source Object-Oriented Library for Discrete-Event Simulation in Java. International Journal of Simulation and Process Modelling, Vol. 4, N. 1, p.69-87.

Swain, J. J. (2007). Discrete Event Simulation Software: New Frontiers in Simulation, OR/MS Today - INFORMS, Vol. 34, No. 5, pp.32-43.

Zuben, F. J. V. Uma caricatura funcional de redes neurais artificiais. Learning and Nonlinear Models - Revista da Sociedade Brasileira de Redes Neurais, Vol. 1, No. 2, p.- 66-76, 2003.