SOBRAPO

PODes
Revista Eletrônica Pesquisa Operacional para o Desenvolvimento

Rio de Janeiro, v.5, n.3, p. 373-405, setembro a dezembro de 2013

# URURAU - UM AMBIENTE PARA DESENVOLVIMENTO DE MODELOS DE SIMULAÇÃO A EVENTOS DISCRETOS

Túlio Almeida Peixoto [a], João José de Assis Rangel [a*], Ítalo de Oliveira Matias [a], José Arnaldo Barra Montevechi [b], Rafael de Carvalho Miranda [b]

[a] *Universidade Candido Mendes(UCAM-Campos), Campos dos Goytacazes - RJ, Brasil*
[b] *Universidade Federal de Itajubá (UNIFEI), Itajubá - MG, Brasil*

## Resumo

O objetivo deste trabalho é descrever o ambiente de desenvolvimento de modelos de simulação a eventos discretos Ururau. A simulação discreta é normalmente utilizada para analisar problemas de pesquisa operacional relacionados principalmente à logística e às linhas de produção, tanto em empresas como no meio acadêmico. Os ambientes de simulação a eventos discretos são os softwares utilizados para a construção dos modelos de simulação. Estes softwares são, na maioria das vezes, de código fechado e o funcionamento interno é como uma caixa preta para os pesquisadores e desenvolvedores dos modelos de simulação. Assim, com a finalidade de explorar a arquitetura interna dos ambientes de simulação discreta, foi proposto o Ururau. O Ururau é um software de código aberto (licença GPL - General Public License) e multiplataforma, pois é desenvolvido em linguagem Java. Ele possui, ainda, uma interface gráfica, que permite o desenvolvimento de modelos de maneira facilitada, através de módulos que podem ser interligados para representar a lógica de um sistema dinâmico e estocástico. A simbologia utilizada para desenvolver os modelos de simulação é baseada em IDEF-SIM, que também é usada para a documentação de modelos conceituais para qualquer outro ambiente de simulação discreta. Os resultados preliminares mostraram que o Ururau permite a execução dos modelos de simulação até cinco vezes mais rápida do que os softwares Arena® ou ProModel® e com resultados equivalentes.

Palavras-chave: Simulação a eventos discretos, simuladores, Java, IDEF-SIM.

## Abstract

The aim of this paper is to describe the Ururau, a development environment of discrete event simulation models. The discrete simulation is commonly used to analyze problems of operational research primarily related to logistics and production lines, both in business and in academia. The discrete event simulation environments are the software used for the construction of simulation models. This software is, in most cases, of closed source and the internal functioning is like a black box for researchers and developers of simulation models. Thus, Ururau was proposed in order to explore the internal architecture of the discrete simulation environment. The Ururau is an open source software (GPL - General Public License) and multiplatform, because it is developed in Java language. It even has a graphical interface, that allows the development of models, in an easy way, via modules that can be linked together to represent the logic of a dynamic and stochastic system. The symbology used to develop simulation models is based on IDEF-SIM, which is also used for documentation of conceptual models for any other discrete simulation environment. Preliminary results showed that the Ururau allows the execution of simulation models up to five times faster than the Arena® or ProModel® software and with equivalent results.

Key words: Discrete event simulation, simulators, Java, IDEF-SIM.

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

Versão inicial submetida em 13/08/2012. Versão final recebida em 23/05/2013.

## 1. Introdução

O objetivo deste trabalho é descrever o ambiente de desenvolvimento de modelos de simulação a eventos discretos Ururau, apresentado, inicialmente, de forma resumida em Peixoto, Rangel e Matias (2011). São detalhados, também, neste trabalho, os principais componentes para funcionamento de um ambiente de simulação a eventos discretos orientado a processos (*Process View*). Além disso, são descritos os principais comandos e a arquitetura do software e, também, demonstradas algumas aplicações básicas do simulador com problemas típicos de sistemas a eventos discretos.

De uma forma geral, os ambientes de desenvolvimento de modelos de simulação a eventos discretos podem ser definidos como ferramentas computacionais que auxiliam a análise de problemas, que vão de logística e manufatura até redes de computadores, sem recorrer a modelos analíticos de Teoria das Filas (Banks *et al*. 2010). Existem conhecidos mais de sessenta simuladores comerciais, conforme pesquisa do Instituto de Pesquisa Operacional e Ciências da Administração - INFORMS (SWAIN, 2007). No Brasil, anteriormente ao Ururau, não se conhece citação de nenhum outro ambiente de simulação a eventos discretos que tenha sido desenvolvido com o propósito de construir modelos de simulação, tanto em interface gráfica (GUI – *Graphic User Interface*) quanto em uma API – *Application Programming Interface* (de maneira complementar a GUI).

O software Ururau utiliza como base a biblioteca de simulação JSL – *Java Simulation Library*, proposto por Rossetti (2008). O JSL permite a construção de modelos orientados a processo e, quando necessário, a adição de novos comandos de processo, quando o modelo de simulação fica mais complexo. Utiliza, também, a linguagem Java, que é multiplataforma e tem licença GPL (*General Public License*) para desenvolvimento em código livre. O apêndice I mostra como obter uma cópia e executar o software Ururau.

Assim, a questão central da pesquisa, apresentada neste trabalho, está na descrição da estrutura e na utilização do *Free and Open Source Software* (FOSS), denominado de Ururau. Espera-se, assim, poder facilitar o uso do software Ururau e permitir:

- a construção de componentes para simulação não previstos em simuladores comerciais;
- o teste de novas funcionalidades para facilitar a construção de modelos de simulação;
- o entendimento de como as ferramentas de checagem de modelos dos ambientes comerciais funcionam, como levantado por Schriber e Brunner (2011);
- a elaboração de modelos na própria interface gráfica do software, de maneira facilitada, através de módulos que podem ser interligados para representar a lógica e a dinâmica de um sistema;
- a construção de modelos mais personalizados, podendo, para isso, alterar diretamente o código fonte para atender a aspectos específicos de um sistema sob análise, dentre outros.

## 2. Linguagens, Ambientes e Bibliotecas Utilizadas em Simulações a Eventos Discretos

Um analista de simulação tem, normalmente, quatro alternativas, quando deseja construir um modelo de simulação a eventos discretos. A primeira é trabalhar com uma linguagem de uso geral somente, a segunda é trabalhar com as bibliotecas de simulação, a terceira alternativa é trabalhar com linguagens específicas para simulação e, por último, trabalhar com os ambientes de simulação a eventos discretos (LAW, 2007).

A primeira alternativa, que é de trabalhar com uma linguagem de uso geral como Java, C++, Python, C#, dentre outras, sem uso de bibliotecas específicas, é a mais flexível de todas as opções. Porém, isto exige maior cuidado por parte do programador ou analista, já que é um processo mais suscetível a erros, como ressaltado por Sargent (2010). Isto ocorre porque o programador tem de implementar, no próprio modelo, o mecanismo de avanço de tempo, a

geração de números aleatórios, as distribuições de probabilidades, além de todo o código relativo à lógica e à dinâmica dos sistemas a eventos discretos.

A segunda alternativa é usar as bibliotecas de software específicas para simulação discreta, mas que usam uma linguagem de uso geral para desenvolvimento de aplicativos. Neste caso, são citadas as bibliotecas JSL (utilizada neste trabalho), SimPy (MULLER e VIGNAUX, 2003) ou DSOL (LANG, JACOB e VERBRAEK, 2003). Estas bibliotecas já têm mecanismos de avanço de tempo, geração de números aleatórios e algumas até possuem funções para tratamento dos dados de entrada, o que facilita a construção de modelos pelo programador (KING e HARRISON, 2010).

A terceira alternativa é usar uma linguagem de simulação específica para sistemas a eventos discretos, como Simula (DAHL, MYHRHAUG e NYGAARD, 1970), GPSS (HENRIKSEN e CRAIN, 2000), dentre outras. Com isto, o modelo computacional torna-se mais sucinto e é elaborado em tempo menor. Porém, a curva de aprendizagem do programador normalmente é maior, por ser uma linguagem de domínio restrito, e a integração com bibliotecas existentes pode ser mais difícil para um modelador não especialista na linguagem.

Comparando as segunda e terceira abordagens, pode-se observar que, na segunda, a curva de aprendizagem é mais curta, já que o usuário já conhece uma linguagem de uso mais geral, que normalmente é mais popular e permite integração relativamente fácil com outras bibliotecas já existentes. Por outro lado, a construção de um modelo pode ser mais trabalhosa, por precisar de estruturas que não são específicas do modelo conceitual, mas necessárias para execução da simulação a eventos discretos.

A quarta e última opção é usar um ambiente de simulação a eventos discretos. Neste grupo, podem-se exemplificar softwares como GPSS, Arena®, ProModel®, Simul8® e

FlexSim®, dentre mais de sessenta simuladores comerciais (SWAIN, 2007). Vale destacar que algumas linguagens de simulação, como GPSS ou SIMAN, evoluíram para estes ambientes de simulação, respectivamente para GPSS (atual) e Rockwell Arena®, como descrevem Goldman, Nance e Wilson (2010). Estes ambientes possuem, pelo menos, uma interface gráfica para facilitar a construção de modelos, além do mecanismo de avanço do tempo, gerador de números aleatórios e apresentação de resultados em forma de relatório. Esta abordagem, apesar de agilizar e facilitar a construção dos modelos, não possibilita a integração dos mesmos com bibliotecas provenientes de linguagens de uso geral.

## 3. A Linguagem IDEF-SIM

Kettinger, Teng e Guha (1997) apontam a presença de mais de 100 técnicas de modelagem de processos disponíveis na literatura. No entanto, para Ryan e Heavey (2006), poucas dessas técnicas fornecem o suporte necessário a um projeto de simulação.

Para Law (1991), Robinson (2008) e Chwif e Medina (2010), a etapa de modelagem de processos é, provavelmente, a parte mais importante e também a mais difícil do processo de desenvolvimento e uso de modelos de simulação. Sendo que, para Leal, Almeida e Montevechi (2008) poucas técnicas de modelagem garantem o apoio correto a projetos de simulação.

Destacando a importância do modelo conceitual em projetos de simulação, Leal (2008) propôs uma técnica de modelagem conceitual específica para ambiente de simulação, denominada de IDEF-SIM.

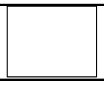
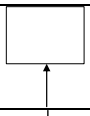
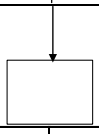
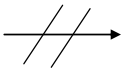
A técnica IDEF-SIM utiliza e adapta elementos lógicos das técnicas de modelagem IDEF0, IDEF3 e fluxograma, permitindo a elaboração de modelos conceituais com informações úteis para a construção do modelo computacional (LEAL *et al.*, 2009).

Segundo Oliveira (2010), a principal característica do IDEF-SIM é a identidade da sua lógica de aplicação com a lógica utilizada em simulação a eventos discretos. Esta característica tem como objetivo criar um modelo conceitual do processo a ser simulado que contenha elementos requeridos na fase de modelagem computacional.

Para Montevechi *et al.* (2010), os elementos utilizados para compor a técnica IDEF-SIM foram selecionados a partir de técnicas de modelagem amplamente utilizadas, como a família de técnicas IDEF (*Integration Definition Language)* e o fluxograma. E embora estas técnicas estejam aptas para a modelagem de sistemas, quando usadas em projetos de simulação, deixam de registrar aspectos importantes, uma vez que não foram estruturadas com foco em projetos de simulação.

A Tabela 1 agrupa os elementos utilizados pela técnica (MONTEVECHI *et al.*, 2010). Leal (2008) apresenta o significado de cada um dos símbolos presentes no IDEF-SIM:

Tabela 1 - Simbologia utilizada na técnica IDEF-SIM.
Adaptado de (LEAL, ALMEIDA e MONTEVECHI, 2008)

| Elementos | Simbologia | Técnica de origem |
|---|---|---|
| Entidade |  | IDEF3 (modo descrição das transições) |
| Funções |  | IDEF0 |
| Fluxo da entidade |  | IDEF0 e IDEF3 |
| Recursos |  | IDEF0 |
| Controles |  | IDEF0 |
| Regras para fluxos paralelos e/ou alternativos |  Regra E<br>  Regra OU<br>  Regra E/OU | IDEF3 |
| Movimentação |  | Fluxograma |
| Informação explicativa |  | IDEF0 e IDEF3 |
| Fluxo de entrada no sistema modelado |  | |
| Ponto final do sistema |  | |
| Conexão com outra figura |  | |

(1) Entidade: são os itens a serem processados pelo sistema, representando matéria-prima, produtos, pessoas, documentos, entre outros. Podem ser agrupadas ou divididas ao longo do processo produtivo e são movimentadas por meios próprios ou por meio de recursos. Uma vez representado no modelo, o símbolo somente aparecerá no momento em que uma nova entidade for criada. Desta forma, torna-se claro o número de entidades a serem utilizadas e em que pontos do modelo as entidades sofrerão uma transformação.

(2) Funções: representam os locais onde as entidades sofrerão alguma ação. Entendem-se como funções: postos de trabalho, esteiras de movimentação, filas e estoques ou postos de atendimento. Estas funções podem modificar uma entidade, como no caso de postos de trabalho, ou mesmo alterar o ritmo desta entidade no fluxo, como uma espera (fila, estoque).

(3) Fluxo da entidade: direcionamento da entidade dentro do modelo, caracterizando os momentos de entrada e saída da entidade nas funções.

(4) Recursos: representam elementos utilizados para movimentar as entidades e executar funções. Os recursos podem representar pessoas ou equipamentos.

(5) Controles: regras utilizadas nas funções, como sequenciamento, regras de filas, programações, entre outros.

(6) Regras para fluxos paralelos e/ou alternativos: estas regras são chamadas de junções, na técnica IDEF3. Dois ou mais caminhos, após uma função, podem ser executados juntos (junção E) ou de forma alternativa (junção OU) ou, ainda, permitindo ambas as regras (junção E/OU).

(7) Movimentação: representa um deslocamento de entidade, no qual o modelador acredita possuir efeito importante sobre o modelo. Ao representar este elemento, espera-se encontrar no modelo computacional uma programação específica para este movimento, como tempo gasto e recurso utilizado.

(8) Informação explicativa: utilizado para inserir no modelo uma explicação, com o objetivo de facilitar o entendimento do mesmo.

(9) Fluxo de entrada no sistema modelado: define a entrada ou a criação das entidades dentro do modelo.

(10) Ponto final do sistema: define o final de um caminho dentro do fluxo modelado.

(11) Conexão com outra figura: utilizada para dividir o modelo em figuras diferentes.

Montevechi *et al.* (2010) destacam ainda que, pelo motivo de a técnica IDEF-SIM ter sido elaborada com foco em projetos de simulação, ela fornece somente informações necessárias à elaboração do modelo computacional, evitando, no caso da modelagem convencional, que um grande volume de informações requeridas pelo modelo computacional não seja contemplado pelo modelo conceitual, obrigando o analista de simulação a buscar estas informações no sistema a ser simulado.

Apesar de recente, a técnica de modelagem IDEF-SIM já foi aplicada em vários casos reais e pode ser vista em vários trabalhos presentes na literatura (COSTA, 2010; COSTA *et al.*, 2010; LEAL, 2008; LEAL *et al.*, 2009; LEAL, ALMEIDA e MONTEVECHI, 2008; MIRANDA, 2012; MONTEVECHI *et al.*, 2010; OLIVEIRA, 2010; OLIVEIRA *et al.*, 2010 e PINHO e MORAIS, 2010).

## 4. Arquitetura e Componentes do Ururau

Na realidade, o Ururau é a continuação do projeto da biblioteca JSL, proposto por Rosseti (2008). Neste trabalho, o autor apresenta a estrutura interna do JSL e indica, como sugestão de trabalho futuro, a construção, dentre outros itens, de uma interface gráfica para dar suporte à construção de modelos de simulação por analistas não especialistas em linguagem Java. Foi, justamente, desta indicação que surgiu a proposta que deu origem ao projeto do Ururau.

A Figura 1 apresenta, então, a arquitetura do software Ururau. Em todas as camadas, foi utilizada a linguagem Java. A biblioteca JSL fica na camada mais inferior e converte o modelo, que é composto por uma série de comandos de processo (*Process View*), para uma seqüência de eventos discretos. A camada intermediária é o núcleo do Ururau, que é composto por comandos de processos específicos do JSL, como criação de entidades, prender (*Seize*), atraso (*Delay*), liberar (*Release*), dentre outros. A camada mais superior trata da conversão do modelo gráfico, que é composto por um grafo dirigido para uma seqüência de comandos do núcleo do Ururau.

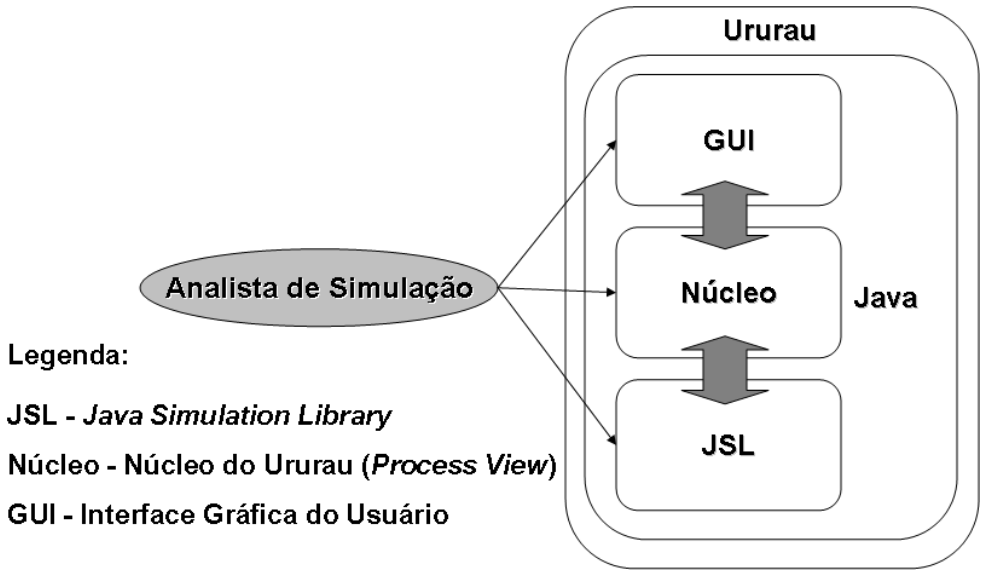



Figura 1 - Arquitetura do Ururau

### *4.1. Comandos do Núcleo do Ururau*

Os comandos do núcleo do Ururau se assemelham ao conjunto de *templates* básicos da maioria dos softwares de simulação. Além disso, o software tem uma interface gráfica, que opera estes comandos que compõem o núcleo do software que, por sua vez, opera o JSL, conforme a Figura 2.

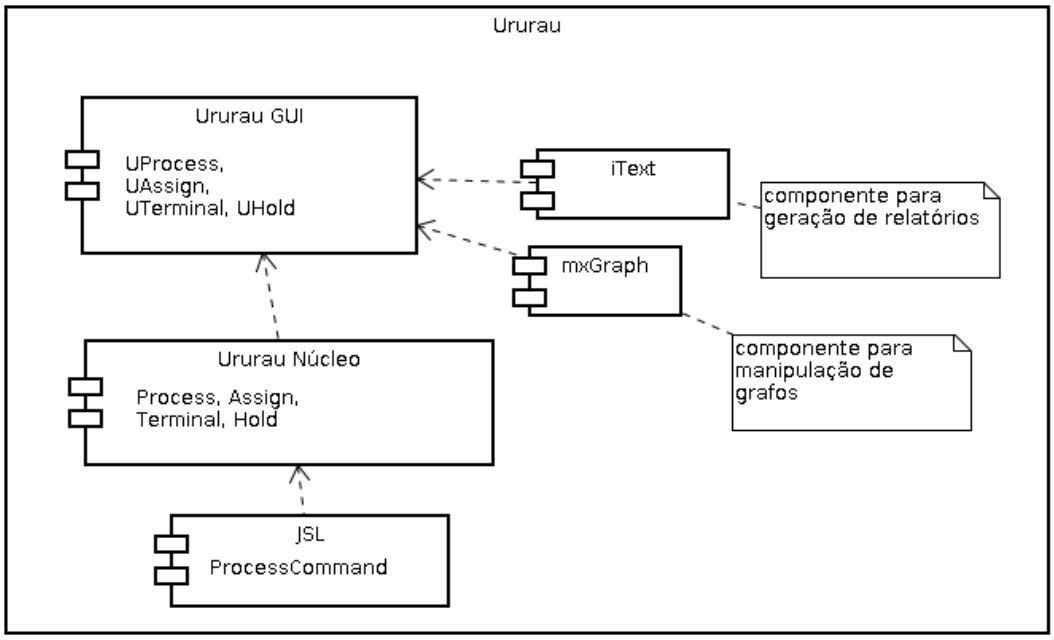



Figura 2 - Diagrama de componentes do Ururau

Cada *template* é composto, internamente, por um ou mais comandos de processo, que estendem as funcionalidades do JSL. O apêndice II apresenta uma breve explicação de como funciona cada comando do software.

### *4.2. Interface Gráfica do Ururau (GUI)*

O propósito da interface gráfica do simulador é facilitar a criação de modelos de simulação. Observe que, na Figura 3, à esquerda, fica o conjunto de funções ou *Templates*. Estes, por sua vez, são montados pelo analista na área de trabalho. Na parte inferior esquerda, é apresentada

uma visão global do modelo, caso este não caiba na tela. A linguagem de representação do modelo é baseada no IDEF-SIM (MONTEVECHI *et al.*, 2010).

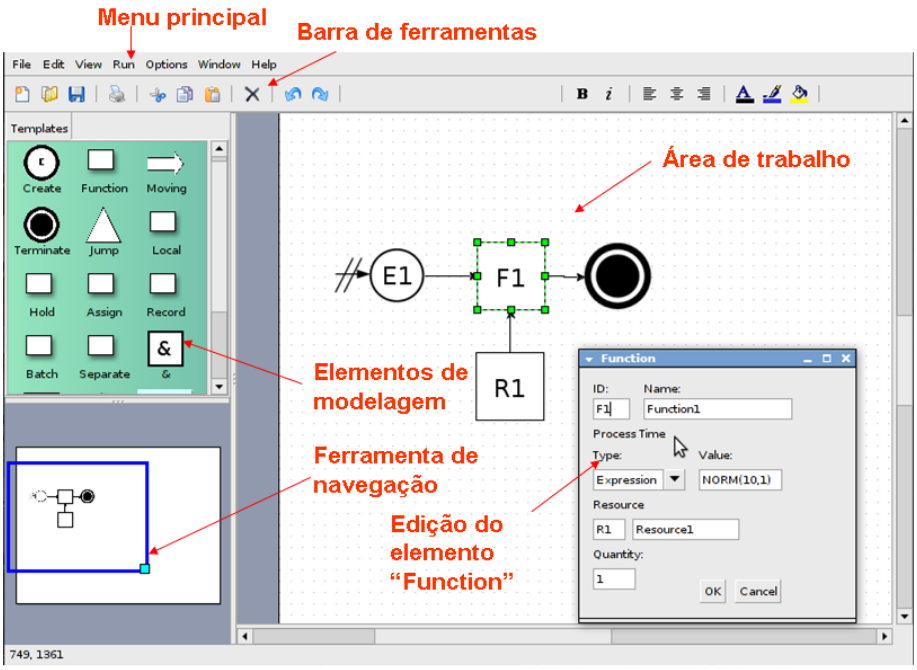
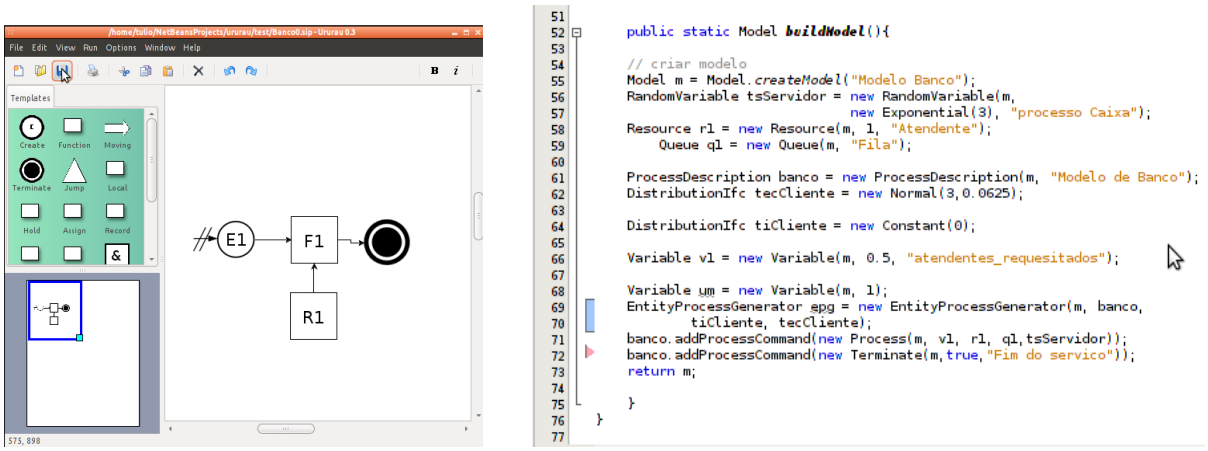



Figura 3 – Ambiente de desenvolvimento do Ururau

Para a construção do modelo de simulação, o modelador precisa apenas arrastar e soltar com o mouse os elementos de modelagem para a área de trabalho, de acordo com a lógica do sistema em análise. A seguir, faz-se a edição do módulo clicando com o botão direito sobre o mesmo para abri-lo. Na caixa de edição, podem-se inserir os dados do modelo como os tempos dos eventos, as regras operacionais do sistema, entre outras funções.

Basicamente, o modelo é um grafo dirigido em que os nós internos são os comandos. Já, os nós externos podem ser geradores de entidades, recursos ou terminais, conforme a Figura 4(a). O modelo equivalente escrito em código Java está mostrado na Figura 4 (b). No entanto, para o desenvolvedor do modelo de simulação em Ururau, o código em Java não é apresentado. Neste caso, o desenvolvedor pode apenas se deter na lógica do sistema a ser modelado.




(a) (b)

Figura 4 – Modelo de simulação em Ururau. (a) Modelo no ambiente de desenvolvimento gráfico do Ururau; e (b) código em JSL do modelo em Ururau

O que a interface gráfica faz, logo ao realizar o comando “Executar Simulação”, é montar o modelo na camada mais baixa do software (Ururau, Java e JSL) e encadear os comandos em linha. A ordem deste encadeamento segue a sequência de visitação do grafo, que é uma busca por profundidade, descrita na Figura 5. A ordem de visitação dos vértices, que são os comandos, começa do comando Decisor (linha 0), vai para Função F1 (linha 1) até ao Terminal (linha 3). Como não existe comando após o Terminal, a ordem de visitação retorna para a linha 0, que é o Decisor, e para na Função F2. Para o último comando visitado, sempre é adicionado um Terminal internamente, fechando, assim, a sequência.

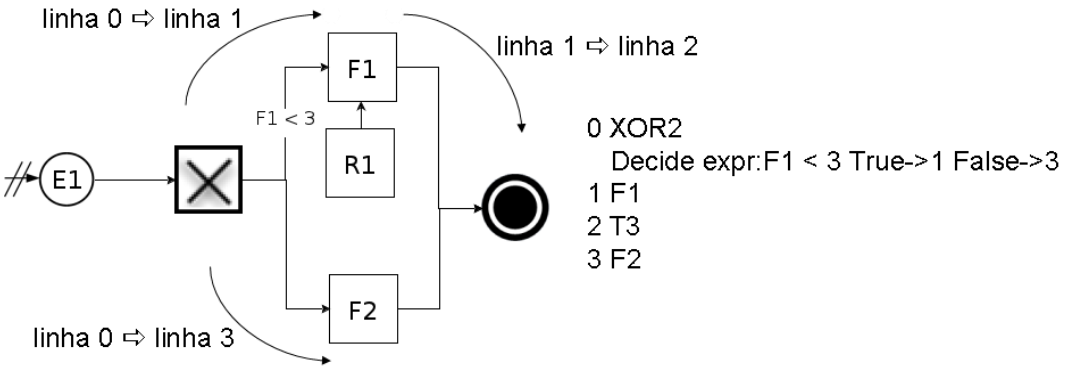



Figura 5 - Busca em profundidade para geração do modelo em código Java

Uma vez desenvolvido o modelo, são definidos os parâmetros da simulação. São eles: o número de replicações, o tempo de replicação e o tempo de aquecimento. A partir daí, o modelo é executado, e, ao final, um relatório com as estatísticas é gerado. Um modelo de relatório é apresentado na Figura 6.

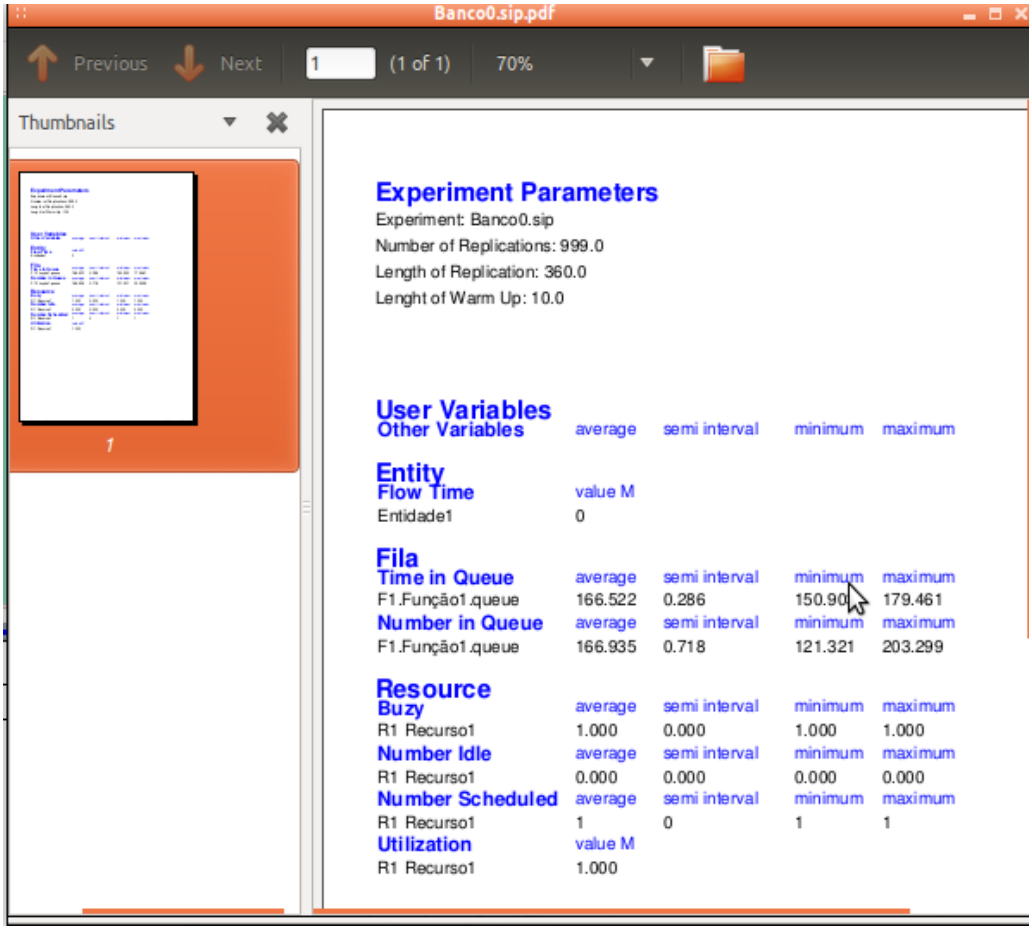



Figura 6 - Relatório com os resultados da simulação

## 5. Aspectos de Modelagem com Ururau

Uma das características de se usar uma descrição do modelo baseada em IDEF-SIM é que esta linguagem já é usada para descrição conceitual dos modelos de simulação e como documentação. Ou seja, a curva de aprendizagem para o modelador é menor, pois, normalmente, ele já conhece o modelo conceitual e a distância entre o modelo conceitual e computacional que, neste caso, certamente é menor.

Outro aspecto é poder elaborar um modelo tanto na camada superior do software (interface gráfica) como na API (núcleo do Ururau). Isto permite que a estrutura de decisão seja feita, por exemplo, com redes neurais (SILVA *et al.*, 2012), em vez das tradicionais expressões lógicas dentro da camada intermediária em código Java.

Outra possibilidade que existe é a de criar componentes de interface gráfica que usam o núcleo do Ururau para, por exemplo, fazer cálculos específicos de um sistema mais complexo, já que existe um modo de criar atributos e expressões dentro do software de maneira customizada, como apresentado em Oliveira *et al.* (2011).

## 6. Experimentos de Simulação com Ururau

Foram realizados dois experimentos com Ururau e comparados com os simuladores de código proprietário Rockwell Arena® 12 e ProModel® 7. Os experimentos foram baseados em exemplos típicos de logística e manufatura. Em Peixoto *et al.* (2012), é apresentada, de forma mais detalhada, uma análise por simulação computacional de um típico sistema a eventos discretos com o ambiente Ururau. Este trabalho pode ser utilizado como um guia para auxiliar a construção de modelos com o software.

Foi utilizada a metodologia proposta por Banks *et al.* (2010) para a construção dos modelos de simulação apresentados a seguir. Os passos utilizados foram: concepção (construção do modelo conceitual; coleta de macro-informações e dados, modelagem dos

dados de entrada); implementação (construção do modelo computacional, verificação e validação); análise (modelo operacional; projeto experimental; experimentação; interpretação e análise estatística dos resultados; documentação e apresentação dos resultados).

Os modelos conceituais foram descritos em IDEF-SIM (MONTEVECHI et al, 2010). Para a verificação e validação do modelo, foi utilizada adicionalmente a metodologia de trabalho proposta por Sargent (2010). Vale ressaltar que os modelos computacionais foram apenas inicializados após a finalização dos modelos conceituais e garantido que todos os pressupostos e hipóteses atribuídos aos sistemas foram corretamente implantados nos respectivos modelos.

Destaca-se que o software Ururau possui ferramenta de teste de aderência e análise estatística de dados semelhante ao Input Analyzer ou o StatFit, que vêm incluídos nos ambientes de desenvolvimento do Arena® e ProModel®, respectivamente. Desta forma, podem-se obter, de maneira semelhante aos softwares comerciais, quando necessário, funções de distribuição de probabilidades aderentes a dados de entradas de modelos de simulação desenvolvidos no Ururau, como descrito em Peixoto *et al.* (2012).

### *6.1. Exemplo 1*

O primeiro exemplo é um sistema básico de suprimento de cana-de-açúcar em usina sucroalcooleira. O problema, descrito na Figura 7, consiste em analisar o corte da cana-de-açúcar na frente de corte (carregamento), o transporte até a usina, o descarregamento e o retorno do caminhão para frente de corte fechando o ciclo.

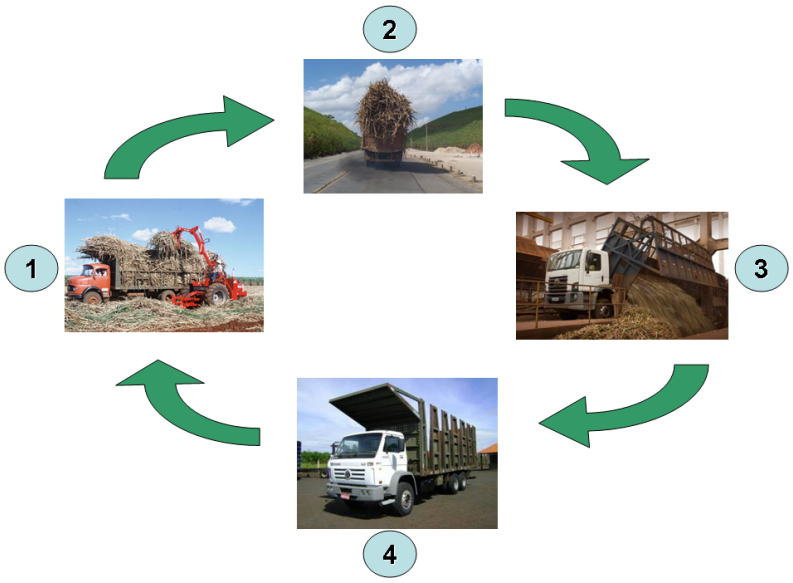



Figura 7 – Estrutura básica do suprimento de cana-de-açúcar em usina sucroalcooleira. (1) carregamento; (2) deslocamento para usina; (3) descarregamento; e (4) retorno para frente de corte

O sistema logístico hipotético levou em consideração o transporte de cana-de-açúcar tipicamente utilizado em usinas sucroalcooleiras e foi adaptado dos trabalhos de Rangel *et al.* (2010) e Iannoni e Morabito (2002).

O modelo conceitual é apresentado na Figura 8. Os caminhões entram no sistema em E1. Note que os retângulos simbolizam as funções de carregamento (F1) e descarregamento (F2). As movimentações são simbolizadas por setas largas (M1 e M2). O triângulo com a letra 'A' simboliza o retorno do caminhão à frente de corte (FC).

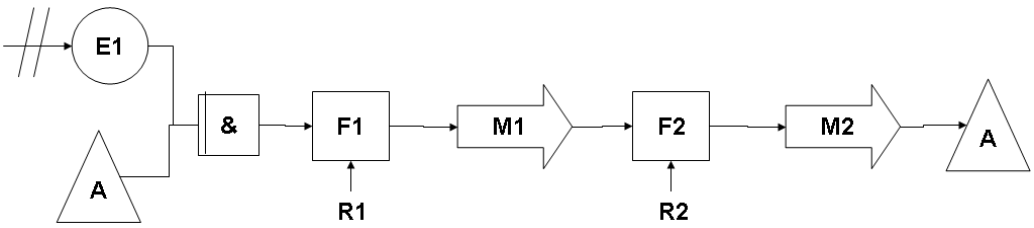



Figura 8 – Modelo conceitual de suprimento de cana-de-açúcar em IDEF-SIM

Faz parte do modelo conceitual, a descrição com os parâmetros mostrados na Tabela 2, que foram adotados para o modelo de modo arbitrário. Note que há 10 caminhões circulando

no sistema, e cada caminhão é carregado paralelamente, de acordo com o número de frentes de corte, neste caso, apenas 1 que é R1. Em seguida, cada caminhão é descarregado com a máquina da usina (R2).

Tabela 2 – Tabela com os parâmetros do modelo de logística

| | Descrição | Parâmetros |
|---|---|---|
| E1 | Entidade: Caminhão | Chegada de 10 caminhões, 1 a cada 10 minutos. |
| F1 | Função: Carregamento | Função Normal (40; 4) minutos. |
| M1 | Movimentação: Ir para usina | Função: Constante de 60 minutos. |
| F2 | Função: Descarregamento | Função: Normal (15; 1,5) minutos |
| M2 | Movimentação: Voltar para a FC | Função: Constante de 60 minutos. |
| R1 | Recurso: Frente de corte | 1 frente |
| R2 | Recurso: Máquina da usina | 1 máquina |

A seguir, na Figura 9, é apresentado o modelo computacional em Ururau. Note a similaridade com o modelo conceitual da Figura 8. A explicação desta similaridade aparece nos itens 3 e 4 do texto.

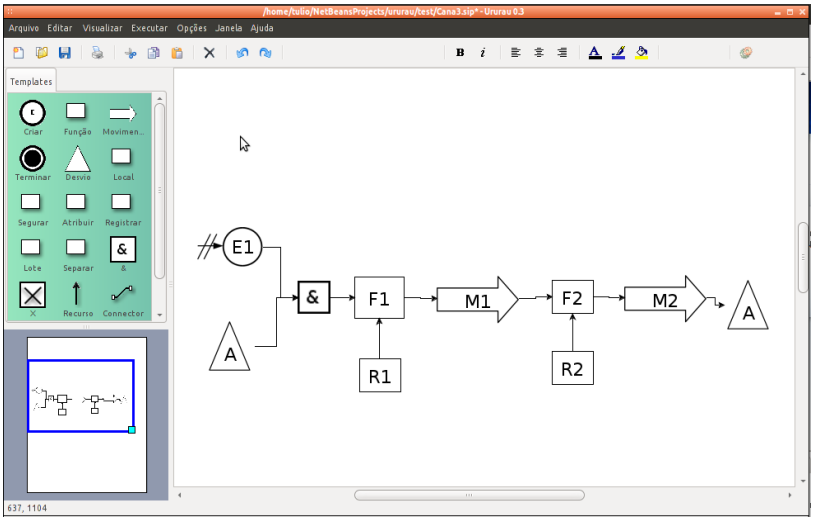



Figura 9 - Modelo do suprimento de cana-de-açúcar no Ururau

A Figura 10 mostra o modelo do mesmo sistema desenvolvido no simulador Arena®. Note que os recursos ficam ocultos no diagrama. Por outro lado, aparece uma barra que simboliza uma fila acima dos processos Carregamento e Descarregamento. A simbologia da

fila indica que estes processos usam recursos, apesar de não explicitar qual recurso é utilizado.

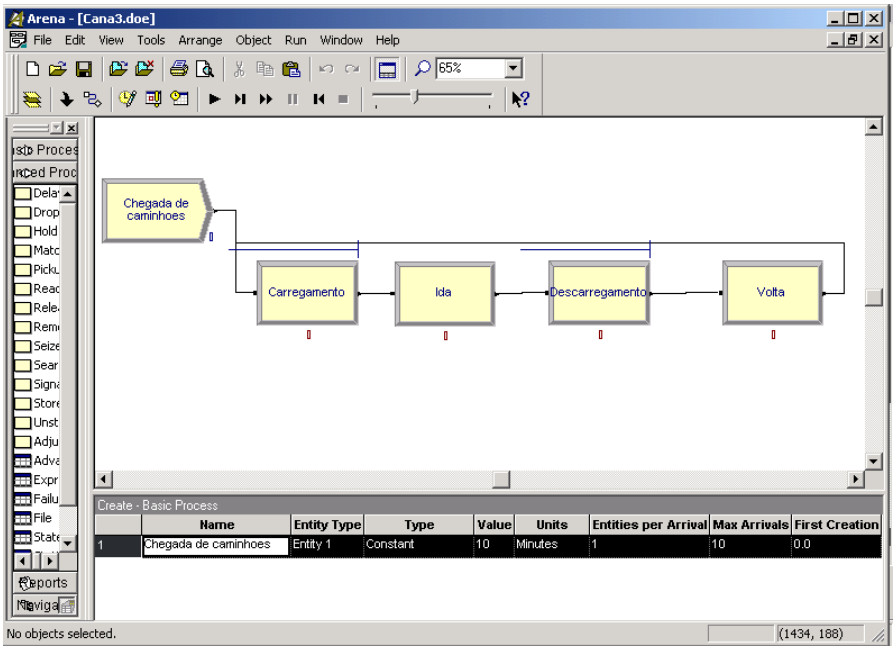



Figura 10 - Modelo do suprimento de cana-de-açucar no Arena®

Já, a Figura 11 mostra o modelo do mesmo sistema em ProModel®. Observe que o modelo desenvolvido em ProModel® difere dos outros dois modelos e não apresenta nenhuma semelhança com a linguagem IDEF-SIM. Esta não é uma vantagem ou desvantagem do software, é apenas uma característica apresentada pelo simulador.

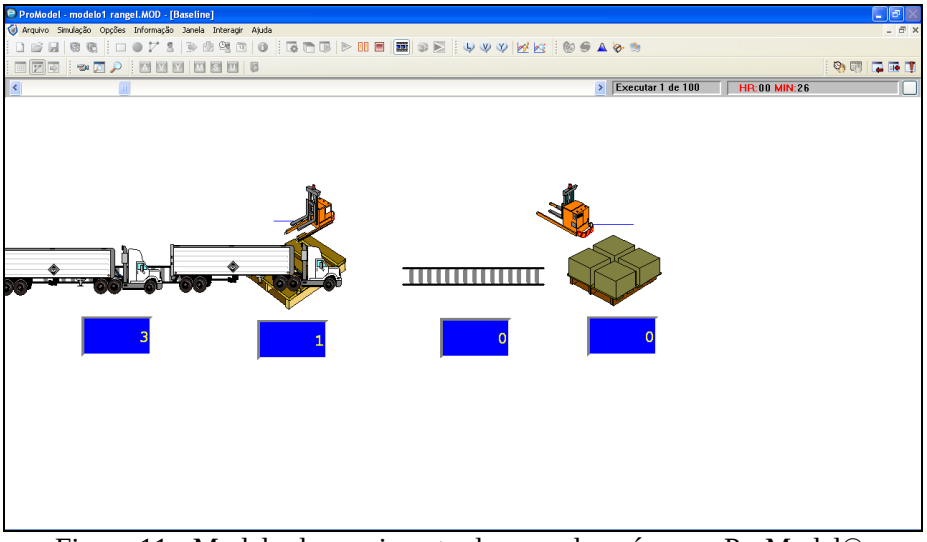



Figura 11 - Modelo do suprimento de cana-de-açúcar no ProModel®.

### *6.2. Exemplo 2*

Foi modelado, também, um problema de manufatura de automóveis, mostrado na Figura 12. Este exemplo foi adaptado a partir de um sistema utilizado em curso de treinamento fornecido pela empresa Paragon. O sistema consiste em analisar o final de uma linha de montagem de automóveis. O sistema é composto por 4 cabines de inspeção e reparo dos produtos finais (R1). Cada cabine tem 1 operador, que realiza todas as operações na cabine. Os veículos (E1) chegam às cabines 1 por vez.

Ao entrar na cabine, é feita uma vistoria inicial. Essa vistoria decide se é necessário fazer uma regulagem (20% dos veículos) ou não. Esta regulagem é realizada com o auxílio de um equipamento eletrônico (R2). Vale ressaltar que há um único equipamento disponível, compartilhado entre as cabines.

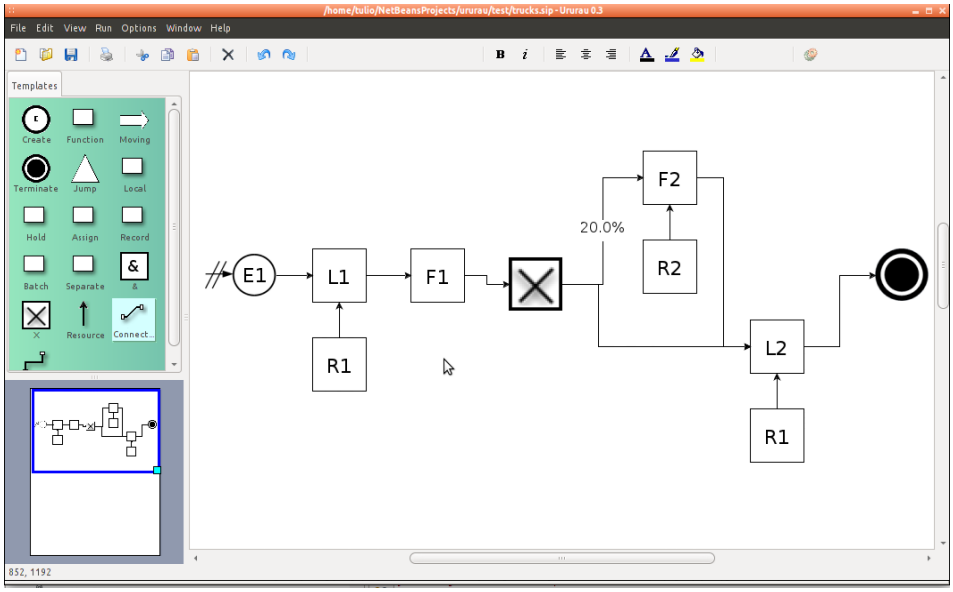



Figura 12 - Final da linha da manufatura de veículos em IDEF-SIM no Ururau

A Tabela 3 descreve os parâmetros do modelo da Figura 12. A capacidade da linha em relação ao número de cabines é 4 e o número de equipamentos eletrônicos é de apenas 1.

Tabela 3 - Parâmetros do modelo de manufatura de veículos

| | Descrição | Parâmetros |
|---|---|---|
| E1 | Entidade: Automóvel | Função: Normal (7; 1) min. 1 carro por vez. |
| L1 | Local: Entrada para as cabines de vistoria | 1 cabine requisitada. |
| F1 | Função: Vistoria | Função: Triangular (2; 5; 9) minutos. |
| F2 | Função: Regulagem | Função: Normal (6; 1) minutos |
| L2 | Local: Saída da cabine | Liberar cabine |
| R1 | Recurso: Cabine | 4 cabines |
| R2 | Recurso: Equipamento eletrônico | 1 equipamento |

Análogo ao exemplo do problema anterior, também foram feitos os modelos em Arena® e ProModel® para fins de comparação, como mostrado nas Figuras 13 e 14.

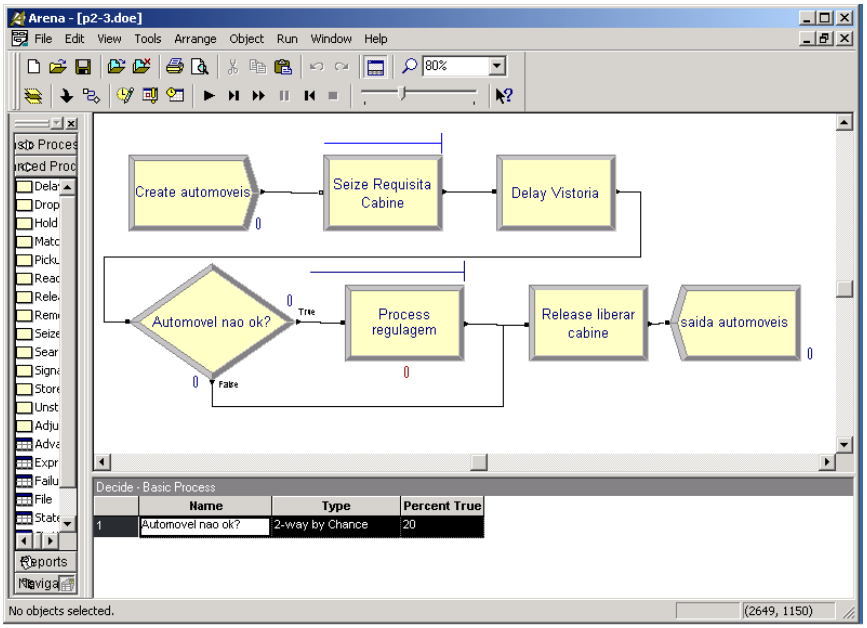



Figura 13 - Modelo em Arena® do final da linha da munufatura de automóveis

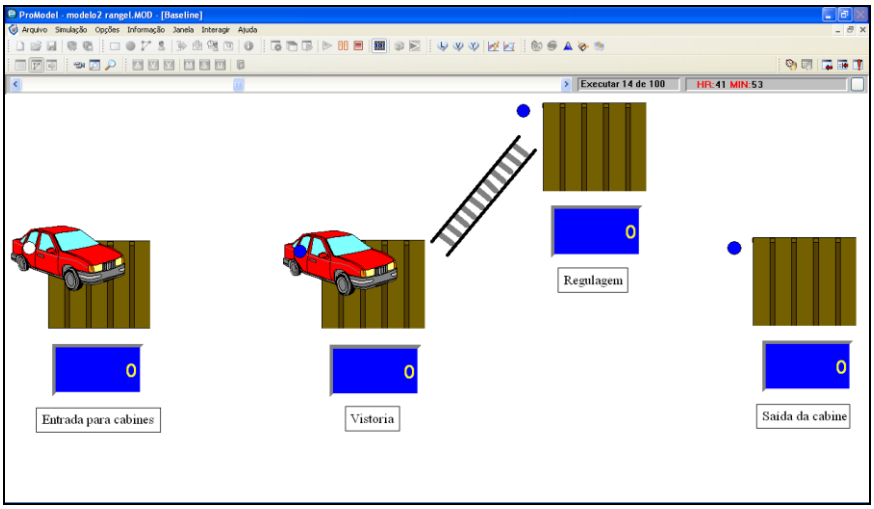



Figura 14 - Modelo em ProModel® do final da linha da manufatura de automóveis

## 7. Avaliação dos Resultados

Os resultados da simulação em Ururau, Arena® e ProModel® para a primeira modelagem, (modelo de suprimento de cana-de-açúcar) são apresentados na Tabela 4. Foram realizadas simulações com 3600 minutos (60 horas), com um período de aquecimento (*Warm-up*) de 60 minutos, até que o sistema entrasse em regime de equilíbrio. Foram realizadas, arbitrariamente, 100 replicações em ambos simuladores. Este número de replicações garante a convergência dos resultados para modelos com estas características. Os resultados comparados são as médias do tempo na fila do carregamento e o número na fila. Estes resultados foram considerados estatisticamente iguais pelo teste de Z com a comparação das duas médias com um nível de confiança de 95%, ou seja, aceita-se a hipótese nula das médias serem iguais.

Tabela 4 - Resultado da simulação Arena® x ProModel® x Ururau para o exemplo 1

| | **Arena®** | | | **ProModel®** | | | **Ururau** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Médio | Semi-intervalo | Desvio padrão | Médio | Semi-intervalo | Desvio padrão | Médio | Semi-intervalo | Desvio padrão |
| Tempo na fila em min. (carregamento) | 221,38 | 0,80 | 4,081 | 220,79 | 0,72 | 3,670 | 219,91 | 0,94 | 4,795 |
| Número na fila em min. (carregamento) | 5,63 | 0,01 | 0,051 | 5,63 | 0,02 | 0,102 | 5,64 | 0,01 | 0,051 |
| Tempo de processamento (em seg.) | 5 | - | - | 7 | - | - | 1 | - | - |

Outro modo de avaliar as respostas entre os simuladores está na Figura 15. Nesta figura, mostram-se a comparação do tempo na fila de carregamento e o número de caminhões na fila de carregamento da cana-de-açúcar extraídos da Tabela 4.

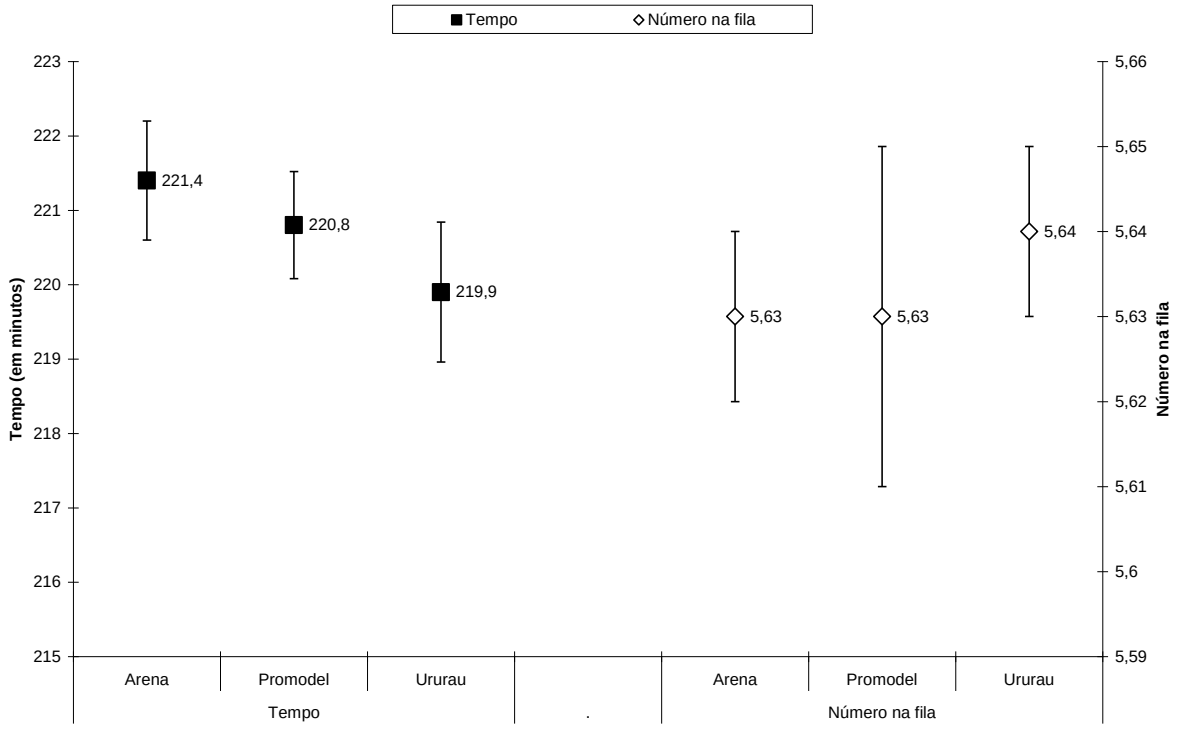



Figura 15 – Tempo na fila e número na fila, comparados entre os simuladores

Outro comparativo da Tabela 4 é o tempo que levou para realizar a simulação. O Ururau foi de cinco a sete vezes mais rápido que os simuladores comerciais (desabilitando as animações). Apesar de a execução ser na ordem de segundos, a vantagem da execução rápida das replicações ajuda na execução de modelos de otimização quando estão juntos aos modelos de simulação.

Os resultados da simulação em Ururau, Arena® e ProModel®, para o segundo modelo (manufatura de automóveis), são apresentados na Tabela 5. Foram realizadas simulações com 3000 minutos (50 horas). Foram realizadas 100 replicações, arbitrariamente, em ambos simuladores. Os resultados comparados são as médias do tempo na fila do carregamento e o número na fila, e estes foram considerados estatisticamente iguais sob 95% de confiança, ou seja, aceita-se a hipótese nula de as médias serem iguais.

Tabela 5 – Resultado da simulação Arena® x Ururau para exemplo 2

| | **Arena®** | | | **ProModel** | | | **Ururau** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Médio | Semi-intervalo | Desvio padrão | Médio | Semi-intervalo | Desvio padrão | Médio | Semi-intervalo | Desvio padrão |
| Tempo na fila em min. (regulagem) | 0,122 | 0,010 | 0,0510 | 0,111 | 0,011 | 0,058 | 0,120 | 0,013 | |
| Número na fila em min. (regulagem) | 0,003 | 0,001 | 0,005 | 0,003 | 0,001 | 0,003 | 0,003 | 0,001 | 0,005 |
| Tempo de processa-mento (em seg.) | 5 | - | - | 7 | - | - | 1 | - | - |

Pode-se comparar, também, a resposta entre os simuladores, na Figura 16. Nesta figura, mostram-se a comparação do tempo na fila de carregamento e o número de caminhões na fila de carregamento da cana extraídos da Tabela 5.

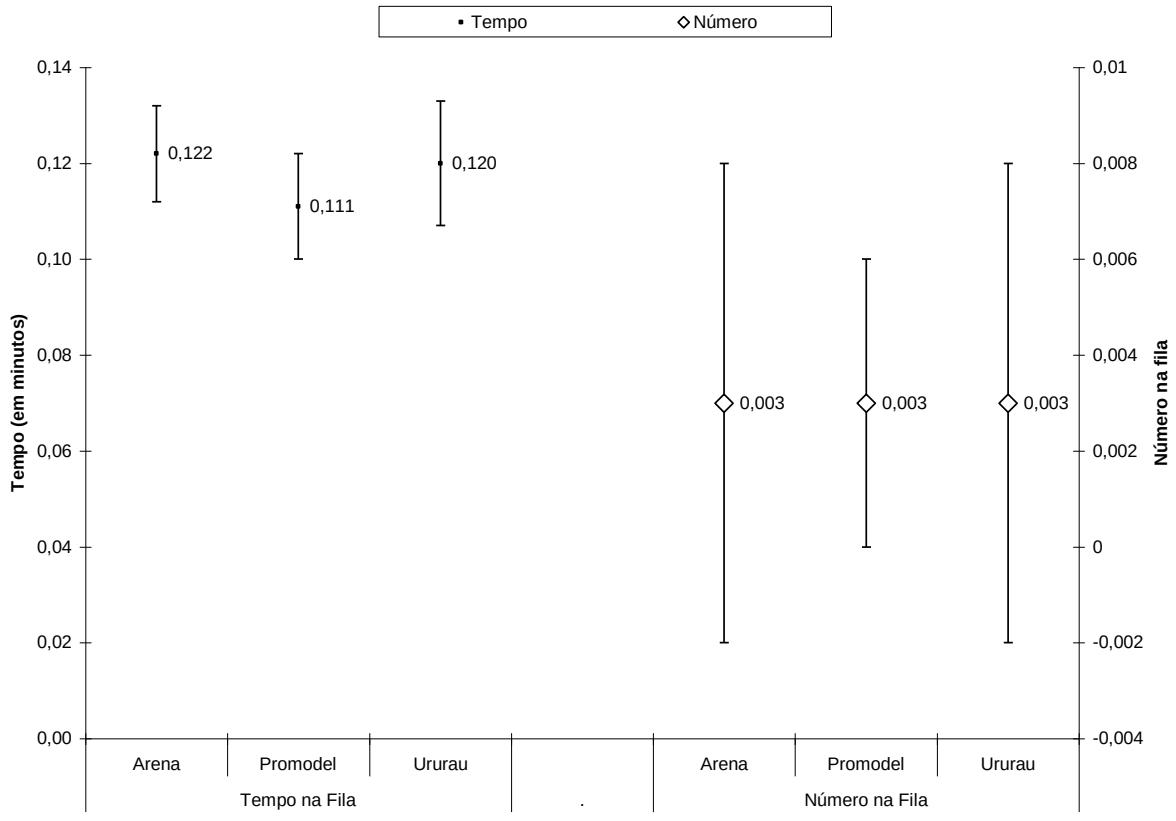



Figura 16 – Tempo na fila e número da fila, comparados entre os simuladores

## 8. Conclusão

Após os testes comparativos entre o simulador Ururau e os simuladores comerciais Arena® e ProModel, pôde-se constatar que todos apresentaram o funcionamento compatível para os modelos executados. Logicamente, não se pode afirmar que o Ururau seja equivalente a estes softwares comerciais. O ambiente de simulação proposto, o Ururau, é um software de simulação que está em construção e pode ser visto como uma espécie de "laboratório de pesquisa e desenvolvimento" na área de simulação a eventos discretos.

No entanto, a utilização do Ururau já pode apresentar certas vantagens. Primeiro, a possibilidade de se entender o código fonte e a estrutura interna de um ambiente de simulação a eventos discretos. Segundo, a possibilidade de se desenvolver modelos de simulação para fins de pesquisa em que se podem adicionar algoritmos específicos ao código fonte do modelo, como um algoritmo de otimização ou de avaliação de decisão por redes neurais. Terceiro, podem-se adicionar novas funcionalidades ao simulador Ururau, de modo a torná-lo mais compatível para algum tipo de abordagem específica e, assim, facilitar a modelagem do sistema desejado.

Outra questão relativa ao uso do Ururau diz respeito à flexibilidade em se poder desenvolver modelos de simulação em qualquer uma das camadas componentes da estrutura do software Ururau. Isto quer dizer que se podem elaborar modelos na camada do JSL (biblioteca de simulação livre), no Núcleo do Ururau ou na camada superior da interface gráfica do Ururau.

Na camada inferior, podem-se construir modelos interligando as bibliotecas em JSL uma nas outras e, a partir daí, obtér-se um modelo de um sistema em código Java. A principal vantagem desta abordagem está na flexibilidade de construção do modelo de simulação.

Como desvantagem, está o período maior para desenvolvimento do modelo e a necessidade de se ter um programador experiente em Java na construção do modelo de simulação.

Na camada intermediária, Núcleo do Ururau, podem-se manipular ainda as bibliotecas JSL e também aproveitar os modelos desenvolvidos para o Ururau. Neste ponto, a principal vantagem está na possibilidade de poder elaborar outros ambientes de simulação com propósitos específicos. Ou seja, caso se deseje criar um ambiente de simulação para simular modelos diversos, dentro de um propósito específico (como apenas a análise de frota de caminhões em usinas sucroalcooleiras), os modelos de simulação poderão ser construídos de forma rápida e precisa, sem que o modelador seja um especialista em simulação discreta.

Destaca-se, finalmente, que a criação do Ururau teve como propósito principal o de contribuir para facilitar a difusão, o uso e a compreensão no Brasil da simulação, desde a sua aplicação prática até a concepção interna de sua estrutura computacional e código fonte. A questão está centrada na formação de pessoas aptas a poderem utilizar a simulação discreta e também os softwares de simulação de forma mais sólida e ampla. Assim, pode-se dizer que as possibilidades de pesquisa e investigações científicas com o Ururau se abrem em várias direções e propósitos, desde a aplicação da ferramenta propriamente dita até o estudo dos seus algoritmos construtivos.

**Agradecimentos**

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

## Referências

Banks, Jerry, Carson Ii, John S., Barry, Nelson L., David M-Nicol. Discrete – Event System Simulation. Fifth Edition, United States of America, Editora: Pearson Education, 2010.

Chwif, L.; Medina, A. C. Modelagem e Simulação de Eventos Discretos: Teoria e Aplicações. 2ª. Ed. São Paulo: Editora dos Autores, 2010.

Costa, R. F. S. Abordagem sistemática para avaliação econômica de cenários para modelos de simulação discreta em manufatura. 139 f. Dissertação (Mestrado em Engenharia de Produção). Universidade Federal de Itajubá (UNIFEI), Itajubá, MG, 2010.

Costa, R.F.S.; Montevechi, J. A. B.; Pamplona, M. S. F.; Medeiros, A. L.; Silva, A. L. F.; Friend, J. D. Discrete-event simulation and activity-based costing to aid the decision making process in a manufacturing cell. In: The International workshop on Applied Modelling & Simulation, Búzios, RJ, 2010.

Dahl, O., Myhrhaug B., Nygaard, K. Common Base Language, Norwegian Computing Center, 1970.

Goldsman, D., R. E. Nance, And J. R. Wilson. A brief history of simulation revised. In Proceedings of the 2010 Winter Simulation Conference, 2010,

Henriksen, J. O., Crain, R. C. GPSS/H: A 23-Year Retrospective View. In: Proceedings of the 2000 Winter Simulation Conference, pp. 177-182, 2000.

Iannoni, A. P.; Morabito, R. Análise do Sistema Logístico de Recepção de Cana-de-açúcar: Um estudo de Caso Utilizando Simulação Discreta. Gestão e Produção (UFSCar), São Carlos, v. 9, n. 2, p. 107-128, 2002.

Kettinger, W.J.; Teng, J.T.C.; Guha, S. Business process change: a study of methodologies, techniques, and tools. MIS Quarterly, v.21, n.1, p.55–80, 1997.

King, D. H.; Harrison, H. S. Discrete-event simulation in Java: a practitioner's experience. Proceedings of the 2010 Conference on Grand Challenges in Modeling & Simulation. Vista, 2010.

Lang. N.A; Jacobs, P. H. M. Verbraek, A. Distributed, Open Simulation Model Development with DSOL Services. In: Proceedings of the 15th European Simulation Symposium and Exhibition, Delft, 2003.

Law, A. Simulation model´s level of detail determines effectivenes. Industrial Engineering, v.23, n.10, p.16-18, 1991.

Law, A. M. Simulation modeling and analysis. 4th edition. McGraw-Hill, 2007

Leal, F. Análise do efeito interativo de falhas em processos de manufatura através de projeto de experimentos simulados. 238 f. Tese (Doutorado em Engenharia Mecânica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2008.

Leal, F.; Almeida, D. A.; Montevechi, J. A. B. Uma Proposta de Técnica de Modelagem Conceitual para a Simulação através de elementos do IDEF. In: Simpósio Brasileiro de Pesquisa Operacional, Anais... João Pessoa, PB, 2008.

Leal, F.; Oliveira, M. L. M.; Almeida, D. A.; Montevechi, J. A. B. Desenvolvimento e aplicação de uma técnica de modelagem conceitual de processos em projetos de simulação: o IDEF-SIM. In: Anais do XXIX Encontro Nacional de Engenharia de Produção, Salvador, BA, 2009.

Miranda, R. C. Algoritmo genético adaptativo para otimização de modelos de simulação a eventos discretos. 149 f. Dissertação (Mestrado em Engenharia de Produção) – Universidade Federal de Itajubá, UNIFEI, Minas Gerais, 2012.

Montevechi, J.A.B.; Leal, F.; Pinho, A.F.; Costa, R.F.S; Oliveira, M.L.M.; Silva, A.L.F. Conceptual modeling in simulation projects by mean adapted IDEF: An application in a Brazilian tech company. In: Winter Simulation Conference, 2010, Baltimore. Proceedings of the 2010 Winter Simulation Conference, 2010. p. 1624-1635, 2010.

Muller K., Vignaux T. SimPy: Simulating Systems in Python. ONLamp.com Python Devcenter, 2003.

Oliveira, G. L. Simulação em Código Livre para Análise do Sistema de Suprimento de Cana-de-açúcar em Usinas Sucroalcooleiras. Trabalho de Conclusão de Curso. (Graduação em Engenharia de Produção) - Universidade Candido Mendes, 2011.

Oliveira, M. L. M. Análise da aplicabilidade da técnica de modelagem IDEF-SIM nas etapas de um projeto de simulação a eventos discretos. 168 f. Dissertação (Mestrado em Engenharia de Produção) – Universidade Federal de Itajubá, UNIFEI, Minas Gerais, 2010.

Oliveira, M. L. M.; Miranda, R. C.; Montevechi, J. A. B.; Leal, F. Desenvolvimento de um projeto de simulação a eventos discretos em uma célula de controle de qualidade de uma empresa de alta tecnologia. In: XLII Simpósio Brasileiro de Pesquisa Operacional, Bento Gonçalves, RS, 2010.

Peixoto, T. A.; Rangel, J. J. A.; Matias, I. O. Ururau - Um Ambiente de Simulação a Eventos Discretos. In: XLIII SBPO. Ubatuba - SP, 2011.

Peixoto, T. A.; Rangel, J. J. A. ; Matias, I. O.; Soares, A. Z. Análise de um Sistema a Eventos Discretos Utilizando um Simulador com Código Livre. In: XVI CLAIO / XLIV SBPO. Rio de Janeiro - RJ, 2012.

Pinho, A. F.; Morais, N. S. Utilização da simulação computacional combinada à técnica de otimização em um processo produtivo. Revista P&D em Engenharia de Produção, v.8, n.2, p.88-101, 2010.

Rangel, J. J. A. ; Cunha, A. P. ; Azevedo, L. R. ; Vianna, D. S. A simulation model to evaluate sugarcane supply systems. In: Proceedings of the 2010 Winter Simulation Conference. Eds.: B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan., 2010, Baltimore, MD - USA. New Jersey: Institute of Electrical and Electronics Engineers, Inc. (WSC - IEEE). p. 2114-2125, 2010.

Robinson, S. Conceptual modelling for simulation Part I: definition and requirements. Journal of the Operational Research Society. v. 59, p. 278-290, 2008.

Rosseti, M.D. Java Simulation Library (JSL): an open-source object-oriented library for discrete-event simulation in Java. In: Int. J. Simulation and Process Modelling, Vol. 4, No. 1, p. 69-87, 2008.

Ryan, J.; Heavey, C. Process modeling for simulation. Computers in Industry, v.57, n.5, p.437-450, 2006.

Sargent, R. G. Verification and validation of simulation models. Proceedings of the 2010 Winter Simulation Conference, IEEE, p. 166-183, 2010.

Schriber, T. J.; Brunner, D. T. Inside Simulation Software: How it works and why it matters. In: Winter Simulation Conference, 2011, Phonix. Proceedings of the 2011 Winter Simulation Conference, 2011.

Silva, D. V. C.; Rangel, J. J. A.; Matias, I. O., Peixoto, T. A. Modelos de simulação a eventos discretos com aspectos de decisões humanas: uma aplicação com Ururau. Revista PODes, 2012.

Swain J. J. INFORMS Simulation software survey. OR/MS Today. Institute for Operations Research and the Management Sciences (INFORMS), USA. Disponível online: http://www.orms-today.org/surveys/Simulation/Simulation1.html. [acessado em 18 de Janeiro de 2012], 2007.

## Apêndice I: Como obter o Ururau

A equipe de desenvolvimento do software Ururau utiliza o controlador de versões para FOSS denominado Bitbucket. O Bitbucket é um site para desenvolvimento colaborativo, ou seja, permite que vários desenvolvedores trabalhem no mesmo código fonte. Assim, o desenvolvedor interessado em cooperar se cadastra e o administrador do projeto permite que esse desenvolvedor submeta as alterações do código para o site. Porém, qualquer pessoa tem acesso ao código sem precisar fazer cadastro. Ou seja, basta instalar o Mercurial (https://mercurial.selenic.com) e fazer o comando "hg clone https://bitbucket.org/ururau/ururau".

O Ururau também pode ser acessado livremente para desenvolvimento de modelos em https://bitbucket.org/ururau. Após baixar o arquivo compactado, descompacte-o em um diretório e execute o arquivo ururau.jar. Lembre-se que é necessário ter instalado no computador o Java Runtime Enviromment (JRE) versão 6.0 ou superior, que pode ser obtido em http://java.com/getjava.

Para construir o modelo usando o código fonte, é preciso um ambiente de desenvolvimento - Integrated Development Environment (IDE), de preferência o NetBeans 7.0, que pode ser obtido em http://www.netbeans.org.

## Apêndice II: Comandos do Ururau

O Ururau é composto, internamente, por um ou mais comandos de processo, que estendem funcionalidades do JSL.

### 1. Gerador de Entidades

*EntityProcessGenerator* (gerador de entidades - Criar). Não é necessariamente um *ProcessCommand*, mas esta classe vem antes de todos os outros comandos no processo de construção do modelo. Ele é responsável pela geração de entidades. A entidade é marcada como visível (por um atributo) e o nível de aninhamento (atributo *nesting*) para uso dos comandos *Batch* e *Separate*. Uma entidade, que é invisível, não é processada por nenhum comando, ou seja, é ignorada.

### 2. Lote

*Batch* (Lote). Este comando faz com que um número determinado de entidades entre em uma fila e seja transformado em uma entidade. O lote pode ser temporário, ou seja, pode ser desfeito posteriormente por um comando que separe lotes (*Separate*) ou pode ser permanente. O funcionamento deste comando é do seguinte modo: as entidades que passam por este comando são marcadas como invisíveis menos a enésima entidade, por exemplo, se o lote é de 10, então 9 são invisíveis e uma é visível. Também sobe um nível de aninhamento, porque pode haver um lote posterior a outro lote.

### 3. Segurar

*Hold* (Segurar). Este comando segura as entidades em uma fila, até que uma condição esteja satisfeita.

**4. Separar**

*Separate* (Separar). Este comando desfaz um lote temporário feito pelo comando Batch. O comando diminui o nível de aninhamento e torna-o visível.

**5. Prender**

*Seize* (Local - Prender). Este comando prende uma ou mais entidades para processamento. Para cada entidade que passa pelo comando *Seize*, um ou mais recursos são ocupados, dependendo da quantidade requisitada de recursos. É possível que uma entidade fique presa enquanto dois ou mais recursos fique "trabalhando" nesta entidade se a quantidade requisitada for maior que um. As entidades que não puderem ser atendidas entram na fila até que o recurso seja liberado.

**6. Liberar**

*Release* (Local - Liberar). Libera o recurso requisitado para a entidade. Basta apontar qual recurso e qual fila para que o recurso utilizado pela entidade corrente seja liberado.

**7. Atraso**

*Delay* (Função - Atraso). Tempo que leva para uma entidade levar para processamento. Recebe uma expressão que é uma distribuição de probabilidade.

**8. Processo**

*Process* (Função - Processo). Agrega em um só comando os comandos *Seize*, *Delay*, *Release* nesta ordem.

### 9. Terminar

*Terminate* (Terminar). Termina a lista de comandos. As entidades visíveis são contabilizadas neste ponto.

### 10. Desvio

*JumpTo* (Desvio). Desvia o processamento de comandos para outro comando. É passado um índice que simboliza a posição de onde está o comando.

### 11. Decisor

*Decide* (X - Decisor). Junto com o *JumpTo*, desvia a execução dos comandos em função de uma condição. Existem dois tipos de decisão: por chance e por condição. Se for por chance, o desvio acontece quando for definido que *o JumpTo* tem *x*% de chance de acontecer para uma posição previamente especificada, se não, o processamento vai para o comando com o ramo falso. Caso o tipo do Decide for por condição, então o *JumpTo* desvia o fluxo caso a expressão lógica especificada for verdadeira. Caso esta expressão seja falsa, o fluxo de entidades segue para o ramo falso.

### 12. Adicionar Atributo

*AddAtribute* (Atribuir - Adicionar atributo). Este comando adiciona um atributo dentro da entidade corrente. O valor do atributo é um número real, que é o resultado da avaliação de uma expressão especificada no comando.

### 13. Adicionar Variável

*AddVariable* (Atribuir - Adicionar variável). Este comando adiciona uma variável no modelo. O valor da variável é um calor real, que é o resultado da avaliação de uma expressão especificada no comando.

### 14. Gravação

*Record* (Gravação). Este comando tem função de contador ou mede o intervalo de tempo em função de um atributo (previamente definido) com o valor do tempo de simulação atual.