UNIVERSIDADE FEDERAL DE SANTA CATARINA

MODELAGEM E SIMULAÇÃO DE SISTEMAS: UMA ABORDAGEM DAS FERRAMENTAS CD++ e ARENA

Antonio Francisco da Rosa Alves

Guilherme Cassol Espíndola

Florianópolis - SC

2012/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

MODELAGEM E SIMULAÇÃO DE SISTEMAS: UMA ABORDAGEM DAS FERRAMENTAS CD++ e ARENA

Antonio Francisco da Rosa Alves

Guilherme Cassol Espíndola

Trabalho de Conclusão de Curso apresentado como Parte dos Requisitos para Obtenção do Grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. João Bosco Mangueira Sobral.

Florianópolis - SC

2012/2

Antonio Francisco da Rosa Alves
Guilherme Cassol Espíndola

MODELAGEM E SIMULAÇÃO DE SISTEMAS: UMA ABORDAGEM DAS FERRAMENTAS CD++ e ARENA

Este Trabalho de Conclusão de Curso foi Julgado Adequado para Obtenção do Título de Bacharel em Sistemas de Informação e Aprovado em sua Forma Final pela Banca.

Florianópolis, 23 de Novembro de 2012.

_____

Prof. Leandro José Komosinski, Dr.

Coordenador do Curso

Banca Examinadora:

_____

Prof. João Bosco Magueira Sobral, Dr.

Orientador

Universidade Federal de Santa Catarina

_____

Prof.ª Carla Merkle Westphall, Dr.ª

Universidade Federal de Santa Catarina

_____

Prof. Renato Bobsin Machado, MSc.

Universidade Federal de Santa Catarina

AGRADECIMENTOS

Dedico este trabalho à minha família, em especial minha mãe Marilda, meu pai Paulo, minhas irmãs Eidy e Mariela, pelo incentivo e apoio nos momentos mais difíceis.

Agradeço ao orientador João Bosco Mangueira Sobral e o coorientador Renato Bobsin Machado, pelo desafio e pela dedicação na orientação deste trabalho.

Ao meu colega e amigo Guilherme Cassol, pelo empenho em realizar este trabalho e por bom convívio neste tempo, tanto nos bons momentos como nos difíceis.

A todos os professores com que tive o prazer de vivenciar e trocar conhecimento no transcorrer do curso.

A todos meus amigos e colegas de classe.

Finalmente, agradeço a todos que me ajudaram nesta empreitada tão importante para atingir esse grande objetivo na minha vida.

(Antonio Francisco da Rosa Alves)

AGRADECIMENTOS

Agradeço a minha família pela compreensão e ajuda no transcorrer do desenvolvimento desse trabalho. Sou muito grato aos meus pais, Jossemar e Lúcia, pelo carinho, amor e apoio dado durante toda minha vida. Agradeço também a minha irmã, Daniela pela amizade e companheirismo.

Agradeço em especial minha namorada Mariana, pelo total apoio em minhas decisões e por compreender meus momentos de ausência.

Agradeço a meu orientador, professor Bosco, por todo o incentivo e suporte para realização desse trabalho e o coorientador Renato, por indicar os principais pontos para desenvolvimento do trabalho.

Agradeço meu amigo Antonio pela parceria no desenvolvimento do trabalho, principalmente por compartilhar suas ideias para esse desenvolvimento.

(Guilherme Cassol Espíndola)

RESUMO

A utilização de sistemas que apresentam alto grau de risco, quanto a sua aplicabilidade real, necessitam de recursos que validem e/ou verifiquem com margem de erro mínima, suas funcionalidades em diferentes situações. Para lidar com esse tipo de problema, ferramentas de modelagem e simulação devem ser utilizadas.

Este trabalho aborda os principais aspectos de modelagem e simulação, fornecendo um guia de como utilizar a ferramenta CD++ e seus diferentes tipos de modelos de simulação: Modelo Acoplado, Modelo Celular e Atômico. Esses aspectos de modelagem são comparados, em linhas gerais aos aspectos de uma simulação usando o simulador ARENA.

Conclusões comparativas aos dos métodos de simulação são mostradas enfatizando a simplicidade de uso, a flexibilidade na modelagem, reuso de modelos existentes e a independência de plataformas, já que CD++ necessita de uma IDE e o ARENA funciona por si só.

Palavras-chave: Modelagem. Simulação. CD++. ARENA.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

DEVS (Discrete Event System Specification) - Especificação de sistemas de eventos discretos.

DEDS (Discrete Event Dynamic Systems) - Eventos discretos de sistemas dinâmicos.

EF (Experimental Frame) - Estrutura Experimental.

IDE (Integrated Development Environment) - Um ambiente integrado para desenvolvimento de software.

FIFO (First In, First Out) - Significa primeiro a entrar, primeiro a sair. Refere-se a uma fila clássica.

LIFO (Last In, First Out) - Último a entrar, primeiro a sair. Refere-se a estruturas de dados do tipo pilha.

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A construção de sistemas computacionais complexos exige do desenvolvedor um grau de conhecimento relativamente avançado, quando se considera métodos formais que envolvem Matemática e Lógica. Tais ferramentas formais se justificam porque precisa-se provar propriedades do sistema e validar níveis de abstração de uma especificação.

Esse grau de conhecimento avançado pode ser minimizado pelo desenvolvedor se este fizer uso de métodos de modelagem e simulação.

De forma geral a simulação deve ser utilizada principalmente em dois momentos; primeiro quando problemas possuem certo grau de complexidade muito elevado para o tratamento analítico; segundo quando a solução para um determinado problema possui alto custo. Com o apoio da simulação, principalmente quando é possível observar características aleatórias, os sistemas podem ser representados com mais detalhes e realismo (FILHO, 1995).

A constante redução do custo de computadores (em conjunto com interfaces gráficas, bibliotecas avançadas, idiomas e outras facilidades), permitiu à simulação tornar-se uma técnica fácil e flexível de usar. Este trabalho aborda e compara os métodos de simulação CD++ e ARENA, através de exemplos didáticos, no sentido de verificar a ferramenta mais adequada para determinado uso.

1.2 PROBLEMA

Tornar um ambiente de simulação de dados, mais preciso e seguro é um dos grandes desafios de um projetista de sistema, pois seu principal objetivo é representar o sistema com alto grau de fidelidade (validação), em relação ao sistema

observado. Com objetivo de avaliar a consistência e determinar as principais funcionalidades de um sistema, se faz necessário o uso de ferramentas específicas que forneçam suporte suficiente para essa representação, garantindo a integridade, fidelidade e segurança das informações desenvolvidas.

O presente trabalho tem o intuito de analisar a seguinte questão problema: como simular um sistema utilizando os principais recursos das ferramentas CD++ e ARENA, traçando um quadro comparativo de suas principais características.

1.3 JUSTIFICATIVA

Muitos sistemas que demandam alto fluxo de informações e realizam operações com alto grau de complexidade, necessitam de um planejamento prévio, que busque realizar os mais diversos levantamentos dos principais detalhes para o desenvolvimento. Neste ponto a simulação deve ser aplicada, com o objetivo de tratar as mais diversas situações que possam abalar o sistema. Espera-se que tal simulação possa ser implementada, utilizando software específico, que modele com alto grau de fidelidade o sistema real.

De acordo com WAINER (2009), modelos de simulação podem ser usados para modelagem e treinamento, pois fornecem soluções rentáveis e sem risco, quando comparados à experimentação. Nas técnicas de simulação, soluções individuais são encontradas para problemas particulares, utilizando um dispositivo (em geral computador), para controlar a compressão e experimentação do tempo.

Desta forma, espera-se com o presente trabalho estudar os principais aspectos sobre a modelagem e simulação de sistemas com as ferramentas CD++ e ARENA.

1.4 OBJETIVOS

Visando responder a questão problema, o trabalho desdobra-se em objetivo geral e objetivos específicos, conforme apresentado a seguir.

1.4.1 Objetivo Geral

Levantar os principais recursos de modelagem e simulação utilizando softwares CD++ e ARENA.

1.4.2 Objetivos Específicos

Através do objetivo geral, apresentam-se os seguintes objetivos específicos:

- Realizar um estudo de modelagem e conceitos de simulação;
- Apresentar os principais tipos de modelos de simulação utilizados pela ferramenta CD++;
- Apresentar as principais funcionalidades da ferramenta de modelagem e simulação CD++ e ARENA;
- Verificar o nível de dificuldade para o usuário, através de exemplos didáticos;
- Comparar as características das ferramentas CD++ e ARENA.

1.5 METODOLOGIA

Segundo Gil (1991) uma pesquisa é um procedimento racional e sistemático que tem como objetivo proporcionar respostas aos problemas propostos. A pesquisa é desenvolvida mediante a aplicação dos conhecimentos disponíveis e a utilização cuidadosa de métodos, técnicas e outros procedimentos científicos.

Para analisar o tema proposto neste trabalho utilizou-se o tipo de pesquisa descritiva, que segundo Gil (1991) visa descrever as características de determinada população ou fenômeno ou o estabelecimento de relações entre variáveis.

O método de pesquisa adotado é a pesquisa experimental. Para obtenção dos resultados será utilizado um software de simulação denominado CD ++. Através de informações sobre modelagem e simulação de sistemas serão analisados dois exemplos de modelagem, utilizando a ferramenta de desenvolvimento CD++, como ambiente de desenvolvimento.

Em relação à abordagem do problema esta pesquisa se configura tanto como qualitativa quanto quantitativa.

Configura-se como pesquisa qualitativa por não ser traduzida em números, mas sim uma abordagem teórica sobre o tema proposto. Richardson (1999, p. 80) menciona que:

> Os estudos que empregam uma metodologia qualitativa podem descrever a complexidade de determinado problema, analisar a interação de certas variáveis, compreender e classificar processos dinâmicos vividos por grupos sociais. Ressalta também que podem contribuir no processo de mudança de determinado grupo e possibilitar, em maior nível de profundidade, o entendimento das particularidades dos comportamentos dos indivíduos.

No que diz respeito à abordagem quantitativa, Richardson (1999, p. 70) menciona que:

> Caracteriza-se pelo emprego de quantificação tanto nas modalidades de coleta de informações, quanto no tratamento delas por meio de técnicas estatísticas, desde as mais simples como percentual, média, desvio padrão, às mais complexas, como coeficiente de correlação, análise de regressão etc.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos relacionados com a proposta deste trabalho. Primeiramente é feita uma descrição global sobre simulação e modelagem. A seguir, são abordados outros temas como: Simulação discreta, modelagem de sistemas dinâmicos de eventos discretos (DEVS), classificação de técnicas de modelagem.

2.1 CONCEITOS DE MODELAGEM E SIMULAÇÃO

2.1.1 Introdução

A primeira técnica usada pelo ser humano para compreender um ambiente é a experimentação, por exemplo, para fazer cerâmica, deve-se misturar água e barro, realizando ensaios diferentes até que a consistência desejada seja obtida. Experimentação era a única forma que o ser humano tinha para aprender sobre o meio ambiente por milhares de anos, e ainda é um dos principais métodos utilizados na resolução de problemas (WAINER, 2009).

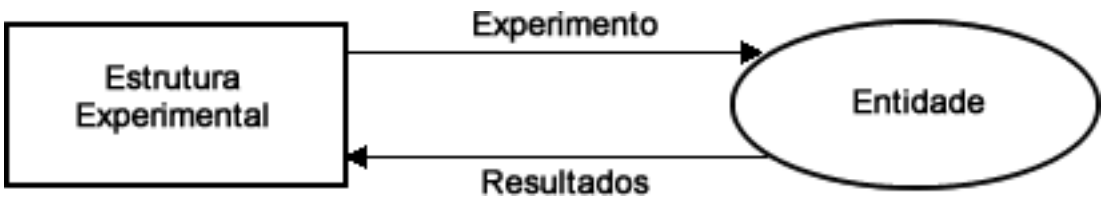

Figura 1: Resolução do problema através de experimentação (WAINER, 2009).

A figura 1 apresenta um esquema básico para experimentação, nela há dois objetos em consideração, um é a entidade em estudo, e o outro a estrutura experimental (EF) que define as condições para tal experimentação. A estrutura experimental define não somente como se devem realizar experimentos na entidade, mas também como se obtém os resultados experimentais. No exemplo citado anteriormente, a entidade é a mistura de água e argila (que pode moldar e, seguidamente solidificar em um forno), e a estrutura experimental é o conjunto de experiências realizadas. Cada experiência seria um ensaio diferente para misturar água e argila (incluindo diferentes quantidades de cada material, temperaturas variadas da mistura, a duração da experiência, entre outros). Os resultados dos experimentos iriam incluir a consistência e a textura esperada, obtida para cada mistura de argila diferente (WAINER, 2009).

2.1.2 Exemplo

Supõe-se a necessidade de estudar a melhor alocação possível de mesas em uma sala de aula, com a finalidade de reduzir os custos energéticos. Então, precisa-se decidir onde colocar as mesas e as fontes de condicionamento de ar. Uma solução de experimentação partiria do princípio de disponibilizar diferentes grupos de estudantes em posições diferentes e utilizaria um sensor (termômetro) para medir a temperatura em diferentes áreas na sala de aula. Neste exemplo, a entidade é a sala de aula e a sua temperatura sob configurações diferentes de mesas. A estrutura experimental (EF) é definida por configurações de estudantes, múltiplos (experimentos) e os tipos de resultados esperados no final de cada experiência. Os resultados são fornecidos pelos termômetros utilizados para medir a temperatura. Como se pode notar, a estrutura experimental permite considerar quais são os objetivos do experimento (temperatura média na sala, o número de alunos, entre outros) e os pressupostos que temos sobre o experimento (por exemplo, deve-se usar um termômetro digital ou um analógico, há interesse em frações de um grau?) (WAINER, 2009).

Em alguns casos, a experimentação não é a solução mais recomendada devido à ética, os riscos (por exemplo, não se pode estudar a propagação de uma epidemia ou de evacuação de incêndio na sala de aula), ou o custo (não se podem estudar todas as configurações possíveis na sala de aula porque agendar um estudo com um grande número de indivíduos por um longo tempo pode ser caro). Em outros casos, a experimentação não é possível de ser realizada (WAINER, 2009).

Os seres humanos têm encontrado formas diferentes para lidar com problemas. Uma delas é abstrair o problema em si e depois raciocinar sobre ele utilizando um modelo do problema. Tomando como base o exemplo 2.1.2 desta seção**,** pode-se começar a usar esta técnica, criando uma imagem mental da sala, dos alunos, dispositivo de aquecimento, etc. Pode-se até começar a pensar sobre a distribuição, dos alunos e mecanismos para melhorar o sistema de aquecimento de acordo com a sua localização, posições de janela, e orientação do edifício. Planejar a melhor distribuição dos elementos da sala de aula (alunos, melhoria do sistema de aquecimento, posição da janela, entre outros.). As ideias podem ser esboçadas no papel, mesmo usando um modelo em escala (WAINER, 2009).

As técnicas de resolução dos problemas são aplicadas de forma analítica, sendo baseada no raciocínio simbólico. A ideia principal é abstrair o que é aprendido sobre a entidade no modelo, que representa exatamente a entidade em estudo. Essa abstração resulta em certa perda de informações, mas permite descrever o comportamento das entidades, analisar e provar suas propriedades no modelo proposto (por exemplo, a controlabilidade e estabilidade no sistema de controle para manter a temperatura da sala) (WAINER, 2009).

Hoje, a modelagem e simulação, com sua abordagem comprovada para a resolução de problemas é beneficiada pelos avanços constantes, com o poder de processamento cada vez mais disponível em menor custo (WAINER, 2009).

A simulação pode ser vista como a manipulação de um modelo apresentando uma visão dinâmica, muito parecida com a realidade. A modelagem trabalha de forma paralela com a simulação, sendo que a modelagem é utilizada na criação de modelos, com a finalidade de representar de forma simplificada e explícita a realidade com algum propósito definido, onde os modelos são manipulados através da simulação de forma dinâmica, com o fluxo de entrada, processamento e saída de algum sistema ou objeto estudado (FILHO, 1995).

Na simulação discreta, os sistemas mudam de estado em instantes discretos (acontecimentos), essas mudanças podem acontecer com incrementos de tempos constantes ou incrementos de tempos variáveis. A variável tempo pode ser contínua ou discreta em determinado modelo, dependendo se as mudanças discretas nas variáveis, ocorrem em qualquer instante de tempo real ou em pontos predeterminados (FILHO, 1995).

Principais aspectos que devem ser levados em consideração quanto ao uso de técnicas de simulação (FILHO, 1995):

- Visualização: a disponibilidade de recursos computacionais para visualização dos resultados obtidos através da simulação (tabelas, diagramas, gráficos), bem como a situação do sistema durante a aplicação de um modelo;
- Custo: deve ser realizado um levantamento do objetivo e da disponibilidade de material suficiente para aplicação, bem como dos recursos físicos (hardware, software, equipamento em geral);
- Interferência: construção de modelos flexíveis para aplicação de mudanças, confrontado com o modelo de um sistema real. Esta característica possui como objetivo, gerar informação suficiente no apoio da definição dos resultados;
- Repetição: modelos de simulação devem ser construídos com objetivo de representar de forma mais fiel possível o sistema proposto, onde sua execução possa ser realizada inúmeras vezes a um custo muito baixo.

As vantagens do uso da simulação são muitas, sendo que algumas destas vantagens merecem ser destacas (WAINER, 2009; FILHO, 2008):

- As decisões podem ser verificadas de forma artificial;
- O mesmo modelo pode ser reutilizado várias vezes;
- Simulações são mais fáceis de criar e usar que várias técnicas analíticas, e precisam de menos simplificações;
- As regras usadas para definir o comportamento do modelo podem ser modificadas facilmente;
- O usuário pode interagir com o simulador, permitindo a análise das interações;

- A entidade original não é afetada pelo estudo, e pode continuar a ser utilizada;
- A identificacação de “gargalos”, que é um fator de grande preocupação no gerenciamento operacional de inúmeros sistemas;
- Hipóteses sobre como ou por que certos eventos acontecem podem ser testadas para confirmação;
- O tempo pode ser controlado, possibilitando reproduzir os fenômenos de maneira lenta ou rápida, para que possa ser mais bem estudados.

2.1.3 Modelagem de Sistemas Dinâmicos de Eventos Discretos (DEVS)

As simulações de sistemas computacionais começaram através do desenvolvimento de técnicas de implementações numéricas, utilizando modelos matemáticos com o objetivo de resolver cálculos de modelos mais complexos de uma forma rápida e precisa. Infelizmente, estes modelos não puderam ser utilizados na simulação da maioria das aplicações computacionais desenvolvidas durante o ano de 1950 em diante. A Revolução Industrial mostrou a necessidade da modelagem de sistemas mais complexos (linhas telefonicas, sistemas de controle de voo, elevadores automatizados, entre outros), que não podem ser adequadamente descritas por equações diferenciais e por aproximação numéricas (WAINER, 2009).

Com a finalidade de resolver esse tipo de problema de uma forma mais eficiente, novas teorias matemáticas (em particular, aquelas fundamentadas na teoria de autômatos) foram aplicadas na análise dos dispositivos automatizados de eventos discretos (CASSANDRAS, 1993; HOPCROFT, MOTWANI, ULLMAN, 2007).

Um exemplo pode ser visto através de uma representação de um modelo de comportamento de um semáforo, com um simples autômato. Esse modelo pode ser utilizado para trabalhar na correção de falhas e na melhoria do comportamento do semáforo (WAINER, 2009).

A figura 2 ilustra o comportamento observado de um semáforo. Inicialmente, a luz é verde por 45 segundos. Então, muda para amarelo durante 10 segundos e finalmente passa para vermelho durante 55 segundos (depois o ciclo é

repetido). A figura 3 representa a modelagem do comportamento de um semáforo através da utilização de autômatos. O autômato (a) representa o modelo de semáforo com o tempo indeterminado para mudança de estado (verde, amarelo, vermelho), já o autômato (b) é uma versão mais complexa, em que ocorre inserção do tempo de alteração de estado no modelo, que permite detectar uma duração errada para um determinado ciclo (WAINER, 2009).

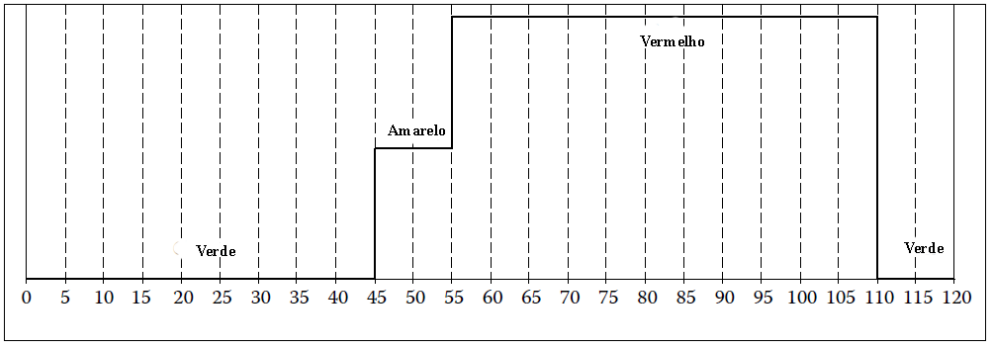

Figura 2: Comportamento de um semáforo (WAINER, 2009).

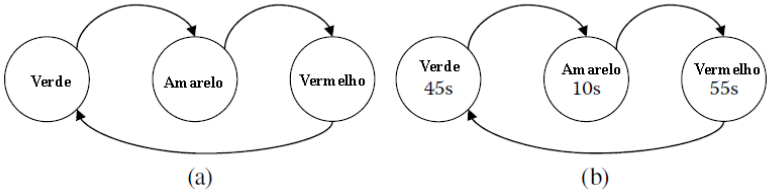

Figura 3: Autômato de duração indeterminado (a) e Autômato simples cronometrado (b), com o comportamento do semáforo (WAINER, 2009).

A figura 4 ilustra o funcionamento de um modelo de semáforo, descrito utilizando a linguagem de programação C-Like. Para t = 0 segundo, a luz é verde, o simulador evolui verificando o atual estado e seu tempo, agindo de acordo com seus valores. Sendo assim, depois de permanecer 45 segundos em verde, o estado é alterado para amarelo, depois de 10 unidades a mais, ele altera para vermelho e finalmente volta para verde. O tempo é incrementado a cada ciclo, utilizando uma

fração de tempo de 5 segundos (que é o máximo divisor dos três períodos, proporcionando maior precisão nos dados capturados) (WAINER, 2009).

```
time = 0; State = Green;
Repeat Forever {
      if (State == Green  AND (time mod 110) == 45)   State = Yellow;
      if (State == Yellow AND (time mod 110) == 55)   State = Red;
      if (State == Red    AND (time mod 110) ==110)   State = Green,
      time = time + 5;
}
```

Figura 4: Implementação do modelo de semáforo baseado no autômato temporizado (WAINER, 2009).

Os modelos de autômatos foram definidos utilizando os valores discretos para representar o tempo e o conjunto de estados. Com a finalidade de verificar algumas das dificuldades, é possível verificar o uso de um semáforo com a utilização de um sensor externo. Sendo esse sensor responsável por determinar o tráfego de veículos nesse local (WAINER, 2009).

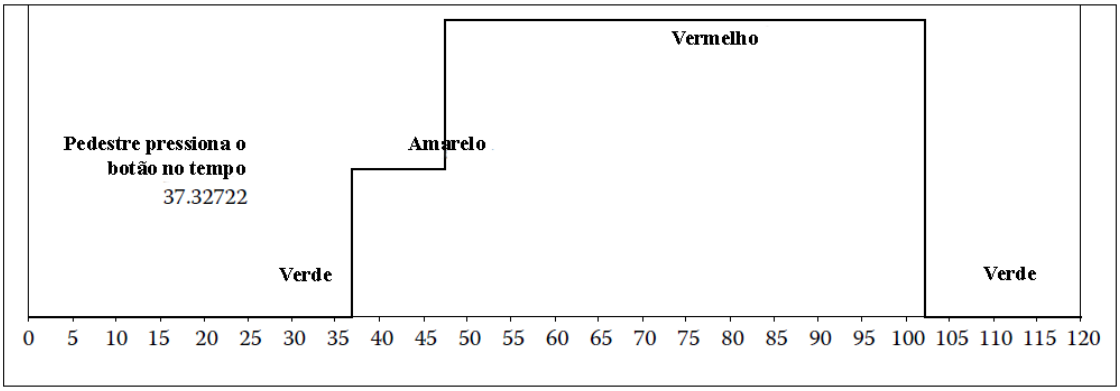

Figura 5: Comportamento de um semáforo com um botão de pedestre (WAINER, 2009).

A figura 5 mostra um exemplo dos dados experimentais para um semáforo. Durante o ciclo de verde (a qual é de 45 segundos de duração), um pedestre chega e pressiona o botão para sua passagem no cruzamento, causando

uma alteração no ciclo. Uma ação tão simples pode apresentar vários problemas como: a modelagem das informações sensoriais externas no autômato. Para ajustar de forma dinâmica os ciclos, sendo que o autômato resultante apresenta ser mais complexo e possui um número maior de estados, a integração deste modelo para um cruzamento complexo, pode crescer exponencialmente (WAINER, 2009).

Este tipo de entidade, que pode ser representada com variáveis discretas e tempo contínuo, são chamados sistemas dinâmicos de eventos discretos (DEVS). Em DEVS, os estados são descritos por trajetórias constantes. DEVS são naturalmente concorrentes e altamente não lineares, sendo difícil de encontrar soluções analíticas genéricas. Os diferentes métodos para modelagem e simulação DEVS são chamados de modelagem e simulação de eventos discretos (WAINER, 2009).

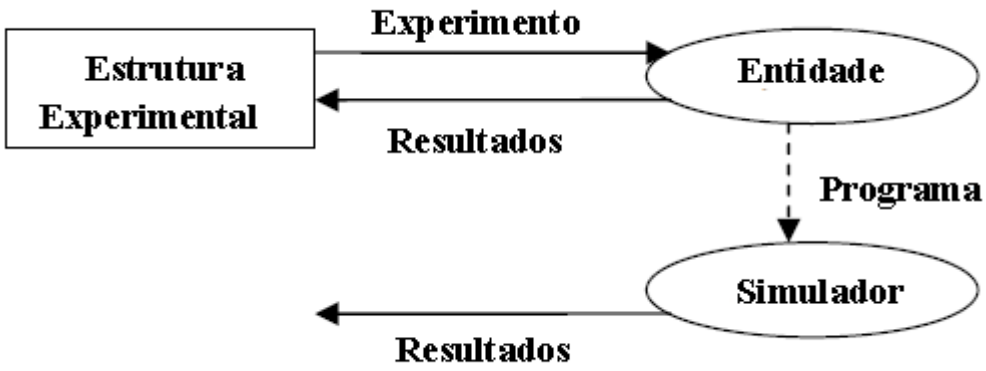

Figura 6: Inclusão de um simulador, que será uma linguagem de simulação (WAINER, 2009).

Para converter o simulador apresentado na figura 6 em um evento discreto, primeiro é necessário um conjunto discreto de variáveis de estado e um relógio, indicando o tempo de simulação atual. Em seguida, é usada uma lista para manter os eventos em ordem cronológica (em geral, armazenados como mensagens) que representam as mudanças de estados do autômato. O tempo em que isso vai acontecer, ti E R, é uma variável contínua. Em cada tempo o simulador irá escolher o primeiro evento na lista, sendo processado e alterado o valor das variáveis de estado e ciclo para o próximo evento (WAINER, 2009).

2.1.4 Classificação de técnicas de Modelagem

A Definição de classificação dos métodos pode ajudar numa melhor compreensão dos dados. A seguinte classificação é fundamentada nos tipos de técnicas utilizadas na modelagem (FISHWICK, 1993).

A **Modelagem conceitual** é uma técnica baseada na criação de um modelo conceitual informal que comunica com a natureza básica do processo. Ele fornece um vocabulário para o aplicativo (que pode ser ambíguo) e uma descrição geral da entidade a ser modelada (WAINER, 2009).

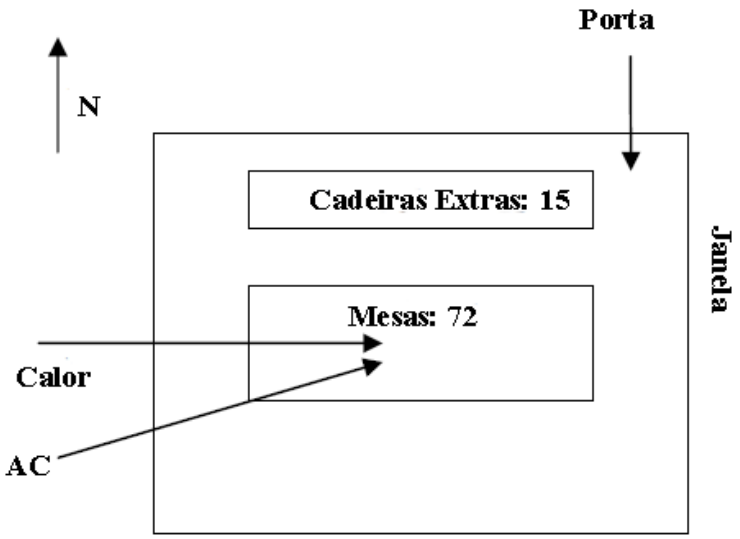

Figura 7: Organização de uma sala de aula (WAINER, 2009).

A figura 7 apresenta um modelo para verificação da temperatura em uma sala de aula durante o dia. A orientação do prédio fica ao norte (N)/sul (S), e há grandes janelas de vidro para o leste. A sala de aula é de 15 × 20 × 3m. Há um total de 72 mesas móveis na sala, que são criadas em oito fileiras de nove mesas cada uma. Há também uma mesa na frente e 15 cadeiras extras disponíveis (sem mesa). A sala tem um projetor, um quadro branco na frente (que está indo para o sul), e um quadro negro na parede oeste. Há três estações de ensino (sem aulas no verão).

Uma vez a cada hora um alarme irá soar para indicar o fim das aulas. Uma vez a cada minuto a temperatura da sala é detectada e o ar condicionado/aquecimento (AC) é ativado quando necessário. A influência da temperatura exterior no resto do edifício é negligenciada. Existe uma fonte de aquecimento/AC no meio da sala (no teto) e uma porta no fundo. Após uma extensa experiência, é verificado que toda vez que uma pessoa chega à sala de aula, a temperatura aumenta 0,05° C, após 5 min. Os parâmetros são: tamanho da sala (15 × 20 × 3m), mesas (72), cadeiras (87), e os quadros. Variáveis de estado são os números de alunos, a atividade (aula/palestra não), as posições das mesas (padrão, círculo, semicírculo), temperatura de aquecimento, temperatura, ar condicionado e estação do ano (verão, outono, inverno) (WAINER, 2009).

A **modelagem declarativa** é uma técnica, que focaliza a evolução do modelo representado como estados (que descrevem o comportamento das entidades em estudo) e as transições entre eles. Um exemplo pode ser representado como um gráfico onde os vértices representam as entidades e os arcos representam a mudanças de estado (transições) (WAINER, 2009).

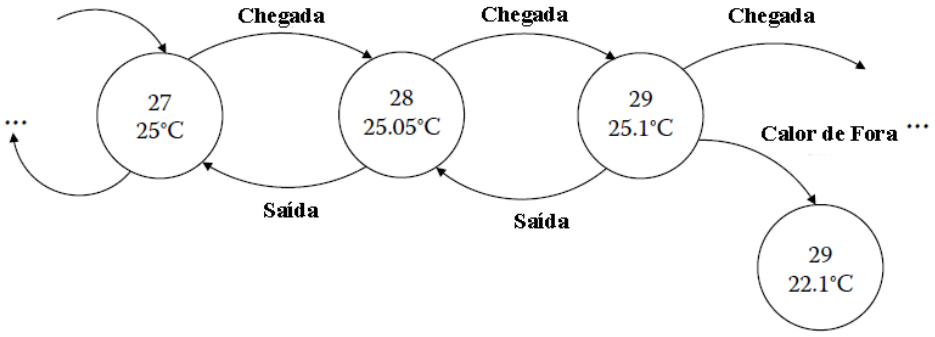

Figura 8: Modelo declarativo do exemplo da sala de aula (WAINER, 2009).

A figura 8 mostra um modelo baseado no estado declarativo representando uma fração do sistema de especificação, em que os alunos chegam ou deixam a sala de aula. A certa altura, tem 27 alunos na sala de aula e uma temperatura de 25 °C. Se um aluno novo chega vamos ter um total de 28 alunos e 25,05° C (isto é um modelo temporal, assim que as informações de temporização

não são incluídas). Se um aluno sai, voltamos ao estado anterior, se outro aluno chega, temos 29 estudantes e temperatura subirá (25,1° C). É também apresentado um estado em que representa o desligamento do aquecimento, o que irá reduzir a temperatura na sala de aula em conformidade. Em eventos com base em modelos declarativos é usada uma notação gráfica baseada com os nós que representam os eventos (as alterações de estado que ocorrem quando há um tipo particular de evento detectado) e as ligações representando as relações entre esses eventos. As mudanças de estado são associadas com cada caso, e as transições podem ter relações lógicas associadas (que define as relações entre os eventos) (WAINER, 2009).

Na **modelagem funcional,** o modelo é definido como uma “caixa preta” e a entrada é um sinal definido ao longo do tempo. A saída depende de uma função interna, e o modelo pode utilizar funções discretas ou contínuas (WAINER, 2009).

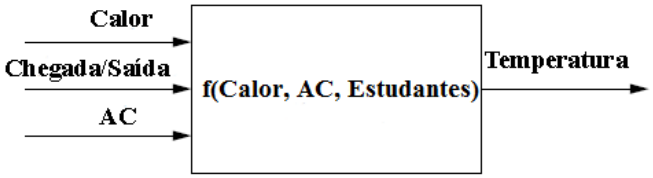

Figura 9: Modelo funcional da sala de aula (WAINER, 2009).

A figura 9 ilustra uma caixa preta que representa a mesma porção de especificação do sistema apresentado nos dois exemplos anteriores. A função f receberá informações sobre os alunos chegando ou saindo da sala de aula. De acordo como número atual de alunos e nível de calor/arcondicionado (AC), ele irá gerar a temperatura da sala (WAINER, 2009).

A **Modelagem espacial** é uma técnica, que noções de espaços são inclusos (isto é, a relação entre o tempo e o espaço é incluída no modelo).

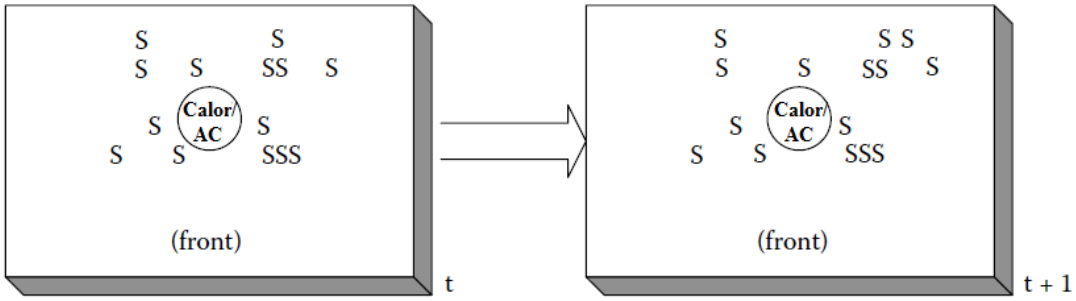

Figura 10: Modelo espacial do exemplo em sala de aula (WAINER, 2009).

A figura 10 apresenta um modelo espacial para o sistema de uma sala de aula. No tempo t, os alunos foram distribuídos na sala de aula como na parte esquerda da figura; no tempo t + 1, um novo aluno já chegou e agora está sentado na fileira de trás. O número de alunos irá influenciar na temperatura da sala de aula (WAINER, 2009).

2.1.5 Algumas definições

De acordo com a teoria de sistemas, um **sistema** pode ser uma entidade natural ou artificial, real ou abstrata, que é uma parte de uma dada realidade restringida por um ambiente. Pode ser visto como um conjunto ordenado de objetos relacionados que evoluem através de diferentes atividades, interagindo para atingir um determinado objetivo. É também chamado de sistema real (ou o sistema de interesse), e é visto como uma fonte de dados de observação, através de uma estrutura experimental (EF) de interesse para o modelador (ZEIGLER, 1976; ZEIGLER, KIM, PRAEHOFER, 2000). Dados observacionais do sistema são usados para definir o comportamento do modelo, que é definido como uma forma específica de dados observáveis num sistema ao longo do tempo dentro de uma estrutura experimental (WAINER, 2009).

Um **modelo** é uma representação compreensível (abstrata e consistente) de um determinado sistema que usamos para entender melhor. Os modelos podem ser construídos de várias formas, e possuem significados diferentes de acordo com

o indivíduo que está fazendo a modelagem. A **estrutura experimental (EF)** define as condições sob as quais um sistema ou um modelo é observado e experimentado, assim, a resolução de problemas está relacionado com a EF dentro do qual o modelo é analisado (ZEIGLER, 1976; ZEIGLER, KIM, PRAEHOFER, 2000).

O processo de pensar e raciocinar sobre um sistema para a descrição abstrata do modelo da realidade é chamada de **modelagem de sistemas**. Pode-se usar o termo paradigma para se referir aos conceitos, leis e mecanismos que são usados para definir um conjunto de modelos. É importante manter uma clara separação entre os sistemas de interesse e os modelos que são usados para pensar sobre eles. (Um modelo é uma representação abstrata do sistema ao invés do próprio sistema, embora seja fácil confundir porque é de costume pensar em modelos de raciocínio sobre sistemas reais) (WAINER, 2009).

A **modelagem de eventos discretos** baseia-se na noção de evento, que é definida como uma mudança no estado do modelo. Um evento ocorre num dado instante (o chamado tempo de evento) fazendo com que o modelo ative uma mudança de estado (por exemplo, pelo menos, um atributo no modelo irá mudar). Finalmente, o estado de um modelo é o conjunto de valores de todos os atributos do modelo num dado instante. Atributos do modelo são normalmente armazenados em variáveis; variáveis de estado são aquelas que irão influenciar a evolução do comportamento do modelo (NANCE, 1981).

A **simulação** pode ser definida como a reprodução do comportamento dinâmico de um sistema de interesse com o objetivo de obter conclusões que serão aplicadas ao sistema (WAINER, 2009).

A fim de ser capaz de criar um modelo e um simulador que represente o sistema com precisão, se faz necessário **verificar** e **validar** as atividades. Usa-se o termo validação para se referir à relação do modelo, o sistema de interesse, e a estrutura experimental. A validação do modelo responde às perguntas e é impossível distinguir o comportamento do modelo e do sistema dentro da estrutura experimental. A verificação é o processo de avaliar se um simulador de um dado modelo apresenta um comportamento desejado (ZEIGLER, 1976; ZEIGLER, KIM, PRAEHOFER, 2000).

2.2 INTRODUÇÃO DEVS

A especificação de sistemas de eventos discretos (DEVS) é uma teoria de modelagem e simulação, que foi definida por B. Zeigler na década de 1970. O formalismo DEVS permite a descrição modular de modelos de eventos discretos que podem ser integrados através de uma abordagem hierárquica (ZEIGLER, 1976; ZEIGLER, KIM, PRAEHOFER, 2000).

De acordo com a teoria DEVS, o sistema de interesse é considerado como uma fonte de dados de comportamento para o estudo dentro de uma estrutura experimental (EF). A estrutura experimental é um conjunto restrito de elementos observados no sistema e as condições sob as quais são observadas. Estes dados são utilizados na criação de uma representação abstrata de um determinado sistema (um modelo). O modelo tenta reproduzir o comportamento do sistema de interesse sob as condições experimentais, através da utilização de um conjundo de instruções, regras ou equações matemáticas. A figura 11 apresenta essas entidades básicas na modelagem e simulação e suas relações (ZEIGLER, 1976; ZEIGLER, KIM, PRAEHOFER, 2000):

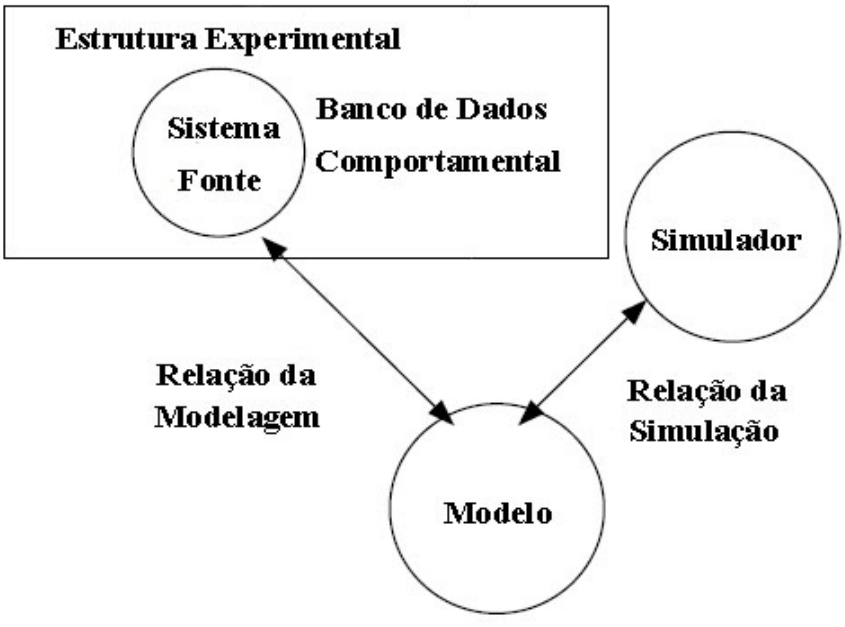

Figura 11: Entidades básicas em uma modelagem e simulação, com seus relacionamentos (ZEIGLER, 1976; ZEIGLER, KIM, PRAEHOFER, 2000).

As especificações formais proporcionam meios para manipulação matemática e permitem a independência da linguagem escolhida para implementar os modelos. O modelo é uma versão simplificada da realidade e sua estrutura. O modelo é construído considerando as condições de experimentação do sistema de interesse, incluindo as condições de trabalho do sistema real e de seu domínio de aplicação. Assim, o modelo é limitado à estrutura experimental em que foi desenvolvido (o que influência a sua construção, as tarefas de experimentação e de validação) (WAINER, 2009).

O modelo é utilizado em outro momento para construir um dispositivo capaz de executar as instruções do modelo (simulador), este gera o comportamento do modelo. Quando se executa as instruções de um modelo, ocorre certa redução na precisão dos resultados obtidos, devido aos recursos limitados numa linguagem de programação e um computador (incluindo a duração da simulação, disponibilidade de memória e precisão das variáveis envolvidas, etc.) (WAINER, 2009).

DEVS foi criado para modelagem e simulação de eventos discretos de sistemas dinâmicos (DEDS), assim, ele define uma maneira de especificar sistemas em que os estados mudam, bem como na recepção de um evento de entrada ou devido à expiração em um determinado período de tempo (WAINER, 2009).

2.2.1 O formalismo DEVS

Um sistema real modelado utilizando DEVS pode ser descrito como uma composição de componentes atômicos e acoplados. Um modelo atômico DEVS é especificado como $\boldsymbol{M} = < \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{s}, \boldsymbol{\delta_{int}}, \boldsymbol{\delta_{ext}}, \boldsymbol{\lambda}, \boldsymbol{ta} >$, onde (ZEIGLER, KIM, 2000):

- $\boldsymbol{X} = \{(p, v) | \, p \in IPorts, v \in X_p\}$ - é o conjunto de eventos de entrada, onde Iports representa o conjunto de portas de entrada e Xp representa o conjunto de valores para as portas de entrada;
- $\boldsymbol{Y} = \{(p, v) | \, p \in OPorts, v \in Y_p\}$ - é o conjunto de eventos de saída, onde OPorts representa o conjunto de portas de saída e Yp representa o conjunto de valores para as portas de saída;

- $\boldsymbol{S}$ - é o conjunto de estados sequenciais;
- $\boldsymbol{\delta_{ext}}: Q\ x\ X\ \rightarrow S$ - é a função de transição externa do Estado, com $Q = \{(s, e)/\ s \in S, e \in [0, ta\ (s)]\}$ sendo que e é o tempo decorrido desde a última transição de estado;
- $\boldsymbol{\delta_{int}}: S\ \rightarrow S$ - é a função de transição interna do estado;
- $\boldsymbol{\lambda}: S\ \rightarrow Y$ - é a função de saída;
- $\boldsymbol{ta}: S\ \rightarrow R0 + U\ \infty$ - é a função de avanço de tempo.

Em qualquer dado momento, um modelo DEVS pode estar em um estado $s\ \in S$. Na ausência de eventos externos, mantém-se nesse estado durante um tempo de vida definido por $ta\ (s)$. Quando $ta\ (s)$ expirar, o modelo gera o valor de $\lambda\ (s)$ através de uma porta y e então muda para um novo estado dado por $\delta_{int}\ (s)$. Uma transição que ocorre devido ao consumo de tempo indicado pela $ta\ (s)$ é chamado uma transição interna. Por outro lado, uma transição externa ocorre devido à recepção de um evento externo. Neste caso, a função de transição externa determina o estado novo, dada por $\delta_{ext}(s, e\ , x)$, onde s é o estado atual, já e é o tempo decorrido desde a última transição, e $x\ \epsilon\ X$ é o evento externo recebido (WAINER, 2009).

A função de avanço de tempo pode assumir qualquer valor real entre $0\ e\ \infty$. Um estado para o qual $ta\ (s) = 0$ é chamado um estado transitório (que irá desencadear uma transição instantânea interna). Em contraste, se $ta\ (s) = 0$ em seguida, s é dito ser um estado passivo, em que o sistema irá permanecer permanentemente a menos que um evento externo é recebido (pode ser usado como uma condição de terminação). Um modelo acoplado DEVS é composto de vários submodelos atômicos ou acoplado. É formalmente definido por $\boldsymbol{CM} = < X, Y, D, \{Md\ |d \in D\}, EIC, EOC, EOC, IC, select >$, onde (WAINER, 2009):

- $\boldsymbol{X} = \{(p, v)|\ p\ \in IPorts, v\ \in X_p\}$ - é o conjunto de eventos de entrada, onde Iports representa o conjunto de portas de entrada e Xp representa o conjunto de valores para as portas de entrada;
- $\boldsymbol{Y} = \{(p, v)|\ p \in OPorts, v\ \in\ Y_p\}$ - é o conjunto de eventos de saída, onde OPorts representa o conjunto de portas de saída e Yp representa o conjunto de valores para as portas de saída;
- $\boldsymbol{D}$ - é o conjunto dos nomes dos componentes e para cada um $d\ \in D$;
- $\boldsymbol{M_d}$ - é um modelo básico DEVS (atômico ou acoplado);

- $\boldsymbol{EIC}$ - é o conjunto de entradas externas de acoplamentos, $EIC \subseteq \{(Self, in_{Self}), (j, in_j)) \mid in_{Self)} \in IPorts, j \in D, in_j \in Iports_j\}$;
- $\boldsymbol{EOC}$ - é o conjunto de saída externas de acoplamentos, $EOC \subseteq \{(i, out_i), (Self, out_{Self})) \mid out_{Self)} \in OPorts, i \in D, out_i \in Oports_i\}$;
- $\boldsymbol{IC}$ - é o conjunto de acoplamentos internos, $IC \subseteq \{(i, out_i), (j, in_j)) \mid i, j \in D, out_i \in OPorts, in_j \in Iports_j\}$;
- $\boldsymbol{select}$ - é a função de desempate, quando $select \subseteq D \to D$, de tal modo que, para qualquer subconjunto não vazio $E, select\ (E) \in E$.

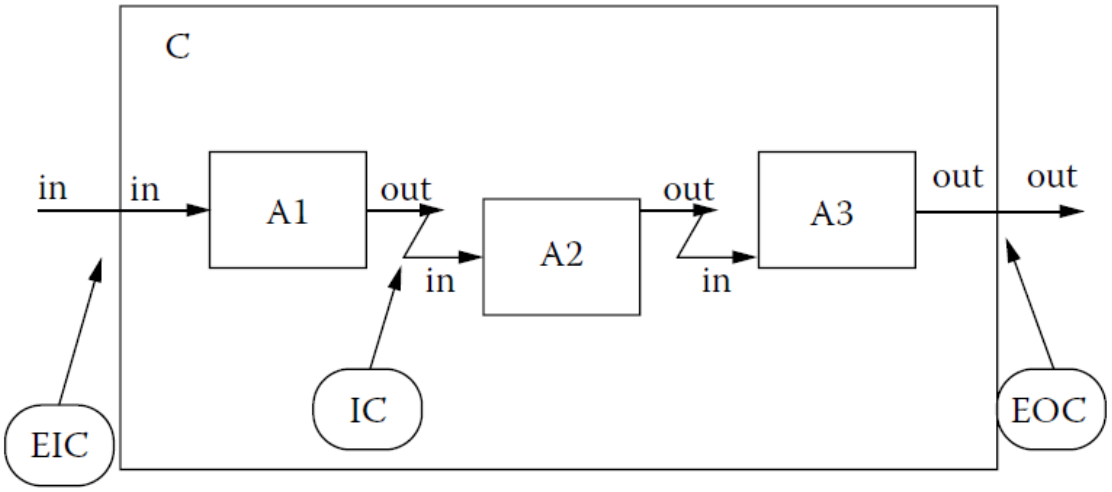

Figura 12: Modelo Acoplado.

A figura 12 apresenta um exemplo de um modelo DEVS juntamente com três subcomponentes, A1- A3. Estes modelos básicos estão interligados através das correspondentes portas de entrada/saída apresentados na figura 12. Os modelos são ligados ao modelo externo acoplado por meio dos conectores de EIC e EOC. O modelo apresentado na figura 12 pode ser formalmente definido como $\boldsymbol{CM} = < X, Y, D, \{Md \mid d \in D\}, EIC, EOC, EOC, IC, select >$, onde (WAINER, 2009):

- $\boldsymbol{X} = \{in, v) \mid in \in IPorts, v \in R\}$;
- $\boldsymbol{Y} = \{out, v) \mid out \in IPorts, v \in R\}$;
- $\boldsymbol{D} = \{A1, A2, A3\}$;

- $\boldsymbol{M_d} = \{M_{A1}, M_{A2}, M_{A3}\}$;
- $\boldsymbol{EIC} \subseteq \{(Self, in), (A1, in))\}; \big(or\ EIC \subseteq \{((C, in), (A1, in))\}\big)$;
- $\boldsymbol{EOC} \subseteq \{((A3, out), (Self, out))\}; \big(orEOC \subseteq \{((A3, out), (C, out))\}\big)$;
- $\boldsymbol{IC} \subseteq \{ ((A1, out), (A2, in)); ((A2, out), (A3, in))\}$;
- $\boldsymbol{select} = \{ A3, A1, A2\}$.

A definição do modelo acoplado apresentado mostra a especificação dos três componentes A1-A3 e seus acoplamentos interno/externo. A propriedade de fechamento assegura que o acoplamento de várias instâncias de classe resulta em um modelo de uma mesma classe, permitindo a construção hierárquica (WAINER, 2009).

3. FERRAMENTA DE MODELAGEM E SIMULAÇÃO CD++

Neste capítulo são abordadas as principais características da ferramenta CD++ (que foi desenvolvida com base na teoria DEVS), abordando seus três tipos de modelos utilizados: **atômico**, **acoplado** e **celular**.

3.1. FERRAMENTA CD++

CD++ é uma ferramenta especializada na Modelagem de Eventos Discretos e simulação com base no formalismo DEVS. Seu funcionamento pode ser tanto em modo autônomo (CPU único) ou paralelo (através de uma interligação de máquinas dentro de uma rede). A ferramenta é utilizada como um plugin no software de desenvolvimento Eclipse, apresentando um agrupamento de ferramentas que devem ser utilizadas na implementação dos projetos, importação e exportação dos dados (WAINER, 2009).

O aspecto essencial do CD++ é seu editor de acoplamento, que fornece a seu usuário a capacidade de manipular os eventos implementados, acoplando arquivos ao modelo criado. Funções básicas do editor são disponibilizadas, bem como comandos de teclado compatíveis. O editor suporta arquivos no formato XML para definição de um tipo especifico de modelo. O sistema foi projetado para fornecer um ambiente que atenda algum dos requisitos de desenvolvimento, como (WAINER, 2009):

- Criar e configurar um projeto;
- Exibir as perspectivas do projeto;
- Editar arquivos do modelo, incluindo palavras reservadas e destaque em palavras chaves; e,
- Apresentar informações e resultados.

Estes requisitos, foram implementados estendendo os pontos de extensão do Eclipse. Os pontos podem agir como uma interface entre o plugin e a

estrutura Eclipse, permitindo que o plugin venha a herdar funcionalidades predefinidas ou interface de usuário a partir do ambiente (WAINER, 2009).

A ferramenta é originalmente projetada para rodar em plataforma Windows, que limita sua utilização, pois devido a variedade de sistemas operacionais existentes no mercado, seu uso pode ser ineficiente em outros sistema operacional. O sistema operacional Linux, pode ser a melhor escolha para utilizar o software, além do Windows, dentre outros fatores positivos como sistema operacional distribuído, portabilidade o uso do compilador GCC como sua principal ferramenta de compilação no Linux é um dos principais fatores. O Compilador GCC é o compilador principal para a compilação dos modelos simulados. GCC possui como significado GNU Compiler Collection, sendo um conjunto de compiladores de linguagens de programação (WAINER, 2009).

Integrado ao Eclipse, CD++ fornece uma interface unificada para todas as tarefas envolvidas na criação e atualização de modelos DEVS e modelos atômicos e acoplados, para análise dos resultados (WAINER, 2009).

3.1.1 DEFINIÇÕES DOS MODELOS

O arquivo que permite definir o modelo (ex: meuModelo.ma) é composto por grupos de definições para os modelos de **acoplamento** e, opcionalmente, a configuração dos modelos **atômicos**. A figura 13 apresenta um exemplo de definição de um modelo acoplado DEVS, cada definição indica o nome do modelo (entre colchetes) e seus atributos. O grupo *[top]* é obrigatório e define o modelo acoplado ao nível superior (RODRÍGUEZ, WAINER, 1999).

```
[top]
components : transducer@Transducer generator@Generator Consumer
Out : out
Link : out@generator arrived@transducer
Link : out@generator in@Consumer
Link : out@Consumer solved@transducer
Link : out@transducer out

[Consumer]
components : queue@Queue processor@Processor
in : in
out : out
Link : in in@queue
Link : out@queue in@processor
Link : out@processor done@queue
Link : out@processor out
```

Figura 13: Exemplo de definição de um modelo de acoplamento DEVS (RODRÍQUEZ, WAINER, 1999).

3.1.1.1 Modelo Acoplado

O modelo acoplado é um agrupamento de modelos atômicos, este modelo especifica como as entradas e saídas dos componentes se conectam, de tal maneira a compor uma hierarquia modular. Esta abordagem permite que o desenvolvedor, ao construir um novo modelo possa reutilizar modelos que já foram validados, assim, facilitando o processo de modelagem (ZEIGLER, 1976; WAINER, 2009).

Nos modelos acoplados existem quatro propriedades para configurar: componentes (utilizando a cláusula "*components*"), portas de saída (cláusula "*out*"), portas de entrada ("*in*") e as ligações entre os modelos (cláusula "*link*"). Estas propriedades são explicadas a seguir (RODRÍGUEZ, WAINER, 1999).

A cláusula **components** descreve os modelos que compõem o modelo acoplado. Se esta cláusula não for especificada, ocorrerá um erro. Exemplo da sintaxe de uso: *nome_modelo@nome_classe* (RODRÍGUEZ, WAINER, 1999).

A cláusula **out** enumera o nome das portas de saída. Esta cláusula é opcional, pois um modelo pode não ter portas deste tipo. A sintaxe para uso é desta cláusula é: *Out porta1 porta2 porta3*. Já a cláusula **in** enumera o nome das portas de entrada. Esta cláusula também é opcional, exemplo da sintaxe desta cláusula: *In porta1 porta2 porta3* (RODRÍGUEZ, WAINER, 1999).

A cláusula **link** descreve o esquema interno e externo de acoplamento. Exemplo de sua sintaxe: *porta_origem[@modelo] porta_destino[@modelo]*. O nome

do modelo é opcional já que, se não for indicado, o modelo acoplado inicialmente definido será usado (RODRÍGUEZ, WAINER, 1999).

3.1.1.2 Modelos Atômicos

Um modelo atômico DEVS é um átomo funcional em um modelo, o qual não pode ser dividido em sub-componentes. Seu comportamento é descrito pela independência de implementação de funções matemáticas e símbolos (ZEIGLER, 1976).

Em determinado momento, se a configuração para os modelos atômicos não é especificada, os valores padrões assumidos pelo desenvolvedor da classe serão utilizados. Na figura 14, a configuração do modelo atômico é especificada (RODRÍGUEZ, WAINER, 1999).

```
[name_of_the_atomic_model]
var_name1 : value1
.
.
.
var_namen : valuen
```

Figura 14: Definição de valores para um modelo atômico DEVS (RODRÍQUEZ, WAINER, 1999).

O nome das variáveis é definido pelo desenvolvedor da classe e deve ser documentado, juntamente com o código fonte. Cada exemplo de um modelo atômico pode ser configurado de forma independente de outros exemplos do mesmo tipo. Na figura 15, duas instâncias da classe *Processor* (derivado de *Atomic*) com configurações diferentes são apresentados (RODRÍGUEZ, WAINER, 1999).

```
[top]
components : Queue@queue Processor1@processor Processor2@processor
.
.


[processor]
distribution : exponential


[processor2]
distribution : poisson


[queue]
preparation : 0:0:0:0
```

Figura 15: Exemplo de definição dos parâmetros para modelos Atômicos DEVS (RODRÍQUEZ, WAINER, 1999).

3.1.1.3 Modelos Celulares

O autômato celular (CA) é basicamente um número infinito de células geometricamente definidas. Cada uma das células possui o mesmo poder computacional, são ligadas a outras células de modo uniforme, e executam as mesmas instruções sincronizadamente e simultaneamente (ZEIGLER, 1976).

O modelo celular (Cell-DEVS) descreve espaços de células como modelos de eventos discretos usando o formalismo DEVS, incluindo funções de atraso para ter uma definição simples do tempo da célula. A proposta do paradigma celular DEVS, considera cada célula de um autômato celular como um modelo de evento discreto, de forma hierárquica e modular. Além disso, o estado da célula é alterado de acordo com uma função local que usa o estado das células presentes e um conjunto finito de células vizinhas (WAINER, GIAMBIASI, 2001).

Modelos celulares são usados com um conjunto de parâmetros de certas características inerentes a eles. Esses parâmetros são (RODRÍGUEZ, WAINER, 1999):

- **Type : CELL:** Indica se o modelo celular é plano ou não (Se não for especificado, será assumido *CELL*). Ele permite definir a quantidade de colunas para um modelo celular unidimensional e bidimensional. A

cláusula *Width* implica necessariamente no uso da cláusula para completar a definição da dimensão. Também permite definir a quantidade de linhas apenas para modelos bidimensionais celulares. Se for necessário definir um modelo unidimensional celular, se deve atribuir o valor 1 a Cláusula de *Height*;

- **Dim: (x0, x1, ..., xn):** Permite definir a dimensão de um modelo celular. Todos os valores *xi* devem ser inteiros. Se a cláusula *Dim* é usada, a invocação das cláusulas *Width* (largura) ou *Height* (altura) na mesma definição do modelo irá produzir um erro.A tupla que define a dimensão do modelo celular deve ter dois ou mais elementos. Isso significa que se for necessário criar um modelo unidimensional celular, deve-se definir uma dimensão (x0, 1);
- **In:** O mesmo que nos modelos acoplados. Esta cláusula pode não ser definida, uma vez que a célula pode não ter portas de entrada conectadas com modelos externos;
- **Out:** O mesmo que nos modelos acoplados. Esta cláusula pode não ser definida, uma vez que a célula pode não ter portas de saída conectadas com modelos externos;
- **Link:** O mesmo que nos modelos acoplados, mas para fazer referência a uma célula que deve ser usado no nome do modelo acoplado em conjunto com (x1, x2,..., xn), sem deixar espaços. Exemplo da sintaxe: *outputPortinputPort @ cellName (x1, x2, ...,xn)*;
- **Border : [WRAPPED | NOWRAPPED]:** Define o tipo de borda para o espaço celular. Por padrão, *NOWRAPPED* é utilizado;
- **Delay : TRASPORT**: Ele especifica o tipo de atraso utilizado em todas as células do modelo;
- **DefaultDelayTime: integer**: Define o atraso utilizado por padrão para os eventos externos (medido em milissegundos). O CD++ não impõe restrições sobre a criação do bairro, permitindo que seja possível utilizar mais do que um vizinho no modelo celular;
- **Initialvalue: Real**: Representa o valor inicial para o espaço da célula. O símbolo representa o valor indefinido;

- **InitialCellsValue: fileName:** Especifica o nome do arquivo que contém os valores iniciais para as células de um modelo celular;
- **InitialCellsValue:** Pode ser usado com qualquer tipo de modelos celulares, mesmo com modelos bidimensionais. Por outro lado *InitialRowValueeInitialRow* não podem ser usados quando a dimensão do modelo é superior a 2. Se a dimensão for 2, qualquer um deles pode ser utilizado, e até uma combinação dos dois, mas neste caso a leitura dos valores no arquivo especificado em *InitialCellsValue* vai substituir os valores da mesmas células definidas por *InitialRowValue* ou *InitialRow*;
- **InitialMapValue : filename:** Especifica o nome do arquivo que contém um mapa de valores que vão ser usados como um estado inicial para um modelo celular;
- **LocalTransition : transitionFunctionName:** Ele indica o nome do grupo que contém as regras que definem o local da função de computação para todas as células;
- **PortInTransition : portName@cellName(xn)transitionFunctName:** Permite definir um comportamento alternativo, quando uma mensagem chega do exterior para a porta de entrada especificado na célula (x1, x2, ..., xn) do modelo celular. Se esta cláusula não é utilizada para uma célula que tem uma porta de entrada, quando chega uma mensagem externa através desta porta, o valor desta mensagem será atribuído à célula usando o atraso especificado pelo defeito na definição do modelo;
- **Zone : transitionFunctionName {range1 [.. Rangen]}:** Permite definir um comportamento alternativo para o grupo de células compreendidas dentro do faixa especificada. Cada faixa é definida como (x1, x2,..., xn) que descreve uma célula única, (x1, x2, ..., xn) .. (y1, y2, yn), descrevendo uma superfície de células, ou uma lista que pode combinar ambas (separação de cada elemento com um espaço em branco). Por exemplo: *Zone: pothole{(10,10) .. (13, 13) (1,3)}*.

A definição das **regras** que descrevem certo comportamento é feita de forma independente para os **modelos celulares** que o utilizam. Isso permite que mais de um modelo celular use a mesma especificação (RODRÍGUEZ, WAINER, 1999).

A linguagem é definida como um novo grupo no interior da especificação, em que cada componente do grupo é uma regra com a seguinte sintaxe: *rule : resultdelay { condition }*.

Cada regra (*rule*) é composta por três elementos: uma condição (*condition*), um atraso (*delay*) e um resultado (*result*). Para calcular o novo valor para o estado da célula, o simulador assume cada regra (na ordem em que eles foram definidos) e, se a condição é avaliada como verdadeira, então o resultado e o seu atraso são avaliados, e estes valores serão designado para a célula. Se a avaliação da condição da regra é falsa, então ele toma a seguinte regra. Se todas as regras são avaliadas sem ter encontrado alguma válida, então a simulação será abortada. Se existir mais do que uma regra válida, leva-se a primeira delas. Se ao avaliar o atraso é obtido o valor indefinido (*undefined*), então a simulação será automaticamente cancelada (RODRÍGUEZ, WAINER, 1999).

3.1.2 Incorporação de Novos Modelos Atômicos

Neste tópico são descritos os mecanismos para definir e incorporar novos modelos atômicos para a ferramenta CD++. No entanto, estes modelos não serão capazes de ser utilizados para criar um modelo celular acoplado, mas podem ser usados para interagir diretamente com outros modelos ou como parte de um modelo DEVS acoplado. Para gerar um novo modelo atômico, deve-se conceber uma nova classe que é derivada da classe *Atomic* e deve acrescentar-se um novo tipo de modelo atômico (Exemplo: *Main Simulator.registerNewAtomics método()*). (RODRÍGUEZ, WAINER, 1999).

Com o novo modelo criado, então é necessário sobrecarregar alguns métodos (*initFunction*, *externalFunction*, *internalFunction*, *outputFunction*). O método **initFunction** é invocado pelo simulador no início da simulação. O objetivo é permitir

a inicialização que o modelo considera necessário. Antes de invocar o método, o valor de sigma é infinito e o estado é passive. Já o método **externalFunction** é invocado quando um evento externo chega a partir de uma porta do modelo (RODRÍGUEZ, WAINER, 1999).

O método **internalFunction** permite definir a função de transição interna. Este método é ativado quando o tempo de simulação é igual ao previsto pela função de avanço de tempo. Já o método **outputFunction** gera saídas, ele é chamado antes a função de transição interna.

Estes métodos (*initFunction*, *ExternalFunction*, *InternalFunction*, *OutputFunction*) podem invocar outros métodos, que permitem interatuar com o simulador (RODRÍGUEZ, WAINER, 1999):

- **holdIn** (*state*, *time*): indica o simulador de que o modelo deve ficar no mesmo estado durante um tempo, e depois que ele irá gerar uma transição interna;
- **passivate** (): indica ao simulador que o modelo entra em modo passivo e que só será reativado quando um evento externo chegar;
- **sendOutput** (*time*, *port*, *value*): envia uma mensagem de saída através da porta;
- **nextChange** (): este método permite obter o tempo restante para a sua mudança de estado seguinte (sigma);
- **lastChange** (): método que permite obter o tempo em que a última mudança de estado ocorreu;
- **state** (): método que obtém o atual estado do modelo;
- **getParameter** (*modelName*, *parameterName*): este método permite acesso aos parâmetros que configuram a classe.

3.1.3 Arquivos de um projeto do CD++

Na ferramenta CD++ ao criar um projeto de modelagem (*CD++ Builder ProjectWizard*), são necessários alguns arquivos para criar uma modelagem e realizar simulações, estes arquivos podem ser ou não opcionais. Seguem abaixo as

extensões e definições das funcionalidades de cada tipo de arquivo (RODRÍQUEZ, WAINER, 1999):

- **FileName.MA:** O arquivo de descrição do modelo;
- **FileName.EV (opcional):** arquivo de eventos externos;
- **Filename.log (opcional):** um arquivo de log gerado pelo simulador;
- **FileName.DRW (opcional):** um arquivo gerado pela ferramenta Drawlog (A ferramenta DrawLog permite representar graficamente a atividade do simulador para modelos celulares em cada instante de tempo, utilizando para isso os dados registrados no arquivo de log), onde os resultados podem ser vistos em uma matriz (disponível apenas em alguns modelos de celulares);
- **FileName.VAL (opcional):** Um arquivo com valores iniciais para o modelo;
- **FileName.BAT**: Um script usado para executar a simulação com os parâmetros corretos (apenas plataformas Windows, mas pode ser simplesmente modificado para ser usado em outras plataformas). Gera um arquivo de log;
- **FileNameDRW.BAT (opcional):** Um script usado para gerar o arquivo drw;
- **NAME.CPP:** O arquivo com código-fonte em C++ que implementa os métodos do modelo atômico. Este arquivo é necessário para recompilar o simulador corretamente;
- **NAME.H:** O arquivo em C++ que é a interface do arquivo name.cpp. Este arquivo é necessário para recompilar o simulador propriamente.

3.1.4 Arquivo para a definição dos valores iniciais do modelo (.VAL)

Para especificar os valores iniciais que um modelo levará pode-se usar a cláusula *InitialCellValue*. Esta cláusula permite especificar o nome de um arquivo que conterá os valores que serão atribuídos por algumas ou todas as células do

modelo antes de começar a simulação. O formato deste arquivo é ilustrado na figura 16 (RODRÍGUEZ, WAINER, 1999).

Figura 16: Formato do arquivo usado para definir os valores iniciais do modelo celular (RODRÍQUEZ, WAINER, 1999).

Por convenção, a extensão *.VAL* é usada no nome deste tipo de arquivo. A dimensão da tupla deve coincidir com o definido para o modelo e deve ser contido no espaço especificado por esta dimensão. Para a definição dos valores iniciais de um modelo celular, um único arquivo deve ser usado, e cada arquivo não será capaz de conter os valores iniciais de dois ou mais modelos. Não é necessário que sejam definidos valores para todas as células do modelo. Essas células que não possuem associação qualquer com valor dentro do arquivo será inicializado com o valor projetado pela cláusula *InitialValue* (RODRÍGUEZ, WAINER, 1999).

A interpretação das linhas do arquivo é realizada em ordem sequencial. Então, se é definido um valor para uma célula e depois um novo valor para a mesma célula, o valor atribuído será o mais recente. Na figura 17 é apresentado o arquivo que descreve os valores iniciais de algumas células de um modelo de quatro dimensões (RODRÍGUEZ, WAINER, 1999).

Figura 17: Exemplo de um arquivo para a definição dos valores iniciais para um modelo celular (RODRÍQUEZ, WAINER, 1999).

3.1.5 Arquivo de mapeamento de Valores Iniciais (.MAP)

Para indicar os valores iniciais para um modelo é possível a utilização da cláusula *InitialMapValue*. Esta cláusula permite especificar o nome de um arquivo que irá conter um mapa de valores que serão atribuídos para as células do modelo antes de começar a simulação. O formato deste arquivo é constituído por uma série de linhas, em que cada um contém um valor real, como é mostrado na figura 18 (RODRÍGUEZ, WAINER, 1999).

```
value_1
...  ...
value_m
```

Figura 18: Formato do arquivo de mapeamento de valores para um modelo celular (RODRÍQUEZ, WAINER, 1999).

Cada valor do mapeamento definido será atribuído a uma célula do modelo de acordo com a ordem em que é mostrado no exemplo que se segue: Suponha-se que há um modelo tridimensional de tamanho celular (2, 3, 2). Em seguida, o primeiro valor do mapa será atribuído à célula (0, 0, 0), o segundo valor para a célula (0, 0, 1), da terceira para a célula (0, 1, 0), o quarto para a célula (0, 1, 1), e assim por diante, até que todas as células do padrão têm atribuído um valor. Se o arquivo que contém o mapeamento de valores não tiver dados suficientes para ser atribuídas a todas as células do modelo, ocorrerá um erro e a simulação será abortada. Por outro lado, se um mapa contém mais valores do que necessário, os valores iniciais serão atribuídos até cobrir os requisitos do modelo, e o restante será ignorado. Por convenção, a extensão *.MAP* é usada no nome deste tipo de arquivo (RODRÍGUEZ, WAINER, 1999).

3.1.6 Arquivo para a definição de eventos externos (.EV)

Os eventos externos são definidos de forma separada para a descrição dos modelos. O arquivo de eventos externos (figura 19) consiste em uma sequência de linhas, onde cada linha descreve o tempo que um determinado evento irá ocorrer, a porta que o evento irá chegar e o valor numérico para o evento (pode ser um número real ou valor indefinido) (RODRÍGUEZ, WAINER, 1999).

```
00:00:10:00 in 1
00:00:15:00 done 1.5
00:00:30:00 in .271
00:00:31:00 in -4.5
00:00:33:10 inPort ?
```

Figura 19: Exemplo de um arquivo para a definição de eventos externos (RODRÍQUEZ, WAINER, 1999).

3.1.7 Formato dos eventos de saída gerados (.OUT)

O arquivo de evento de saída (figura 20) gerado pelo simulador tem o formato parecido com o arquivo de definição de os eventos externos. Neste arquivo é especificado, o tempo que um determinado evento irá ocorrer, a porta de saída, e o valor do elemento de saída (RODRÍGUEZ, WAINER, 1999).

```
00:00:01:00 out 0.000
00:00:02:00 out 1.000
00:00:03:50 outPort ?
00:00:07:31 outPort 5.143
```

Figura 20: Exemplo de um arquivo de saída (RODRÍQUEZ, WAINER, 1999).

3.1.8 Formato do arquivo de log (.LOG)

O arquivo de log (figura 21) registra o fluxo de mensagens entre os modelos que participam na simulação. Cada linha do arquivo mostra o tipo de mensagem, o tempo em que ocorrem, o emissor e o destino. Esta informação é comum para todas as mensagens. Além disso, se a mensagem é de tipo X ou Y, em seguida, ele incluirá a porta e o valor. Para as mensagens do tipo D que vai incluir o tempo do evento seguinte, ou "..." no caso de que este tempo é infinito. Os números que figuram ao lado do nome do simulador associados a cada modelo são apenas para obter informações para o desenvolvedor (RODRÍGUEZ, WAINER, 1999).

```
Mensaje I / 00:00:00:000 / Root(00) para top(01)
Mensaje I / 00:00:00:000 / top(01) para life(02)
Mensaje I / 00:00:00:000 / life(02) para life(0,0,0)(03)
Mensaje I / 00:00:00:000 / life(02) para life(0,0,1)(04)
Mensaje D / 00:00:00:000 / life(0,0,0)(03) / 00:00:00:100 para life(02)
Mensaje D / 00:00:00:000 / life(0,0,1)(04) / 00:00:00:100 para life(02)
Mensaje D / 00:00:00:000 / life(0,0,2)(05) / 00:00:00:100 para life(02)
Mensaje D / 00:00:00:000 / life(0,1,0)(06) / ... para life(02)
Mensaje * / 00:00:00:100 / Root(00) para top(01)
Mensaje * / 00:00:00:100 / top(01) para life(02)
Mensaje * / 00:00:00:100 / life(02) para life(0,0,0)(03)
Mensaje * / 00:00:00:100 / life(02) para life(0,0,1)(04)
Mensaje Y / 00:00:00:100 / life(0,0,0)(03) / out /      0.000 para life(02)
Mensaje D / 00:00:00:100 / life(0,0,0)(03) / ... para life(02)
Mensaje Y / 00:00:00:100 / life(0,0,1)(04) / out /     10.500 para life(02)
Mensaje D / 00:00:00:100 / life(0,0,1)(04) / ... para life(02)
Mensaje X / 00:00:00:100 / life(02) / neighborchange / 0.000 para life(0,0,0)(03)
Mensaje X / 00:00:00:100 / life(02) / neighborchange / 0.000 para life(0,1,0)(06)
Mensaje X / 00:00:00:100 / life(02) / neighborchange / 0.000 para life(0,2,0)(09)
Mensaje X / 00:00:00:100 / life(02) / neighborchange / 0.000 para life(0,9,0)(30)
```

Figura 21: Fragmento de um arquivo de log (RODRÍQUEZ, WAINER, 1999).

3.2 EXEMPLOS DE PROJETOS UTILIZANDO MODELAGEM ACOPLADA, ATÔMICA E CELULAR.

Este capítulo apresenta alguns modelos de simulação desenvolvidos na ferramenta CD++ utilizando as técnicas de modelagem acoplada, atômica e celular.

3.2.1 EXEMPLO DE MODELAGEM ACOPLADA E ATÔMICA, CONSTRUÇÃO DE UMA FILA.

Antes de iniciar a apresentação de um projeto desenvolvido com modelagem acoplada usando a ferramenta CD++, é necessário compreender o conceito de uma FILA. Uma fila é um dispositivo de armazenamento temporário que usa um mecanismo FIFO (*First In First Out*) como estrutura, a figura 22 apresenta a estrutura de uma fila:

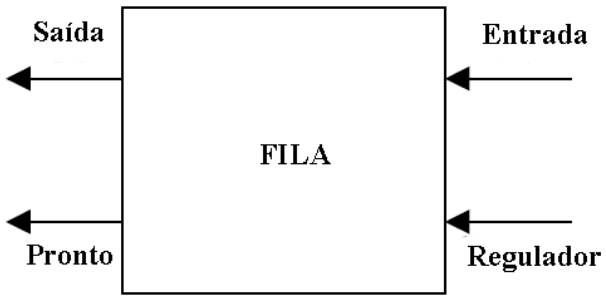

Figura 22: Estrutura de uma Fila (RODRÍQUEZ, WAINER, 1999).

A fila deve ter uma porta de entrada (*in*) que permite ao restante dos elementos inseridos no modelo serem armazenados por fila, e uma porta de saída (*out*) para retornar os valores armazenados. O tempo de atraso entre a chegada do elemento e a sua saída é configurável pelo utilizador. Para cumprir estes requisitos a fila define duas portas, uma porta pronto (*Done*) que indica a recepção do elemento

enviado pela porta de saída (*out*) e de um regulador de fluxo (*stop-send*) (RODRÍGUEZ, WAINER, 1999).

A figura 23 indica o primeiro passo para criar um projeto de modelagem acoplada, sendo necessário ir à opção: *File* > *New* > *Project*. Adiante, é apresentada uma janela como na figura 24, possibilitando escolher o tipo de projeto, que para este caso é o *CD++ Builder Project Wizard*, conseguinte é necessário definir um nome para o projeto, neste exemplo, define-se por Fila.

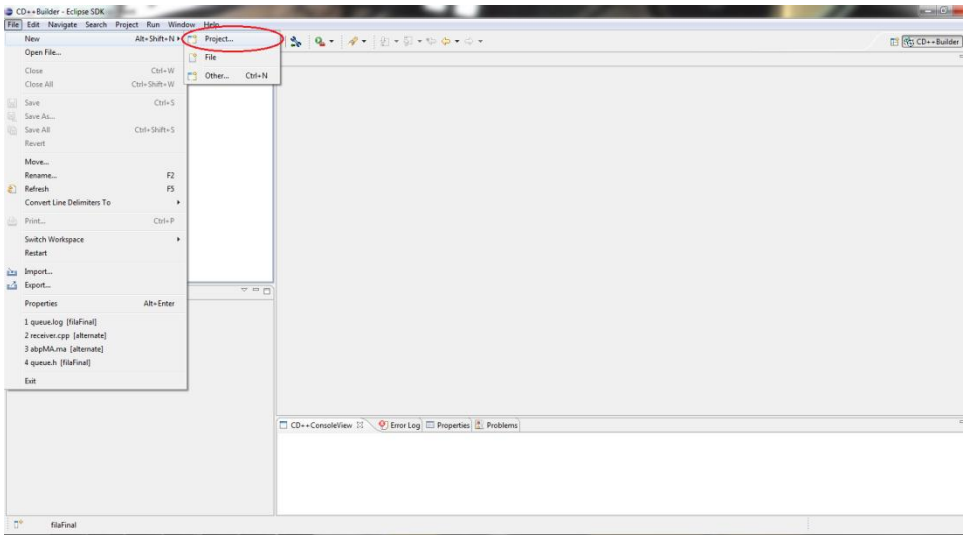

Figura 23: Criando um novo projeto DEVs na ferramenta CD++.

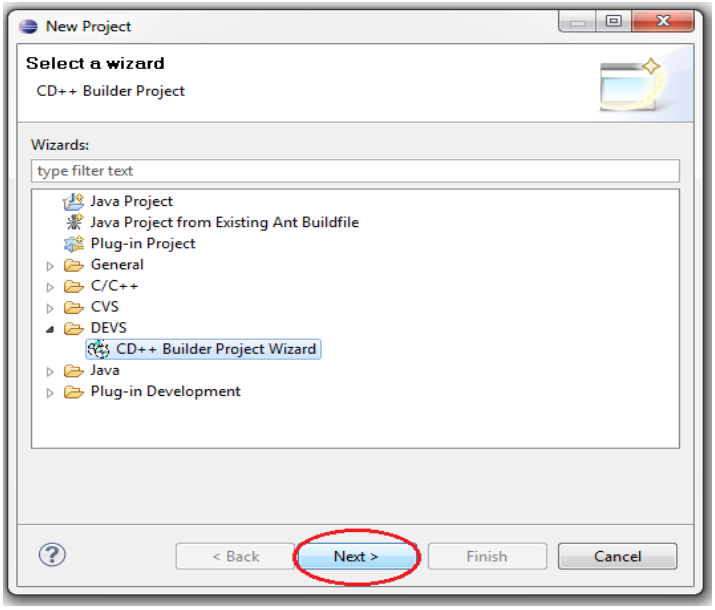

Figura 24: Selecionando um tipo do projeto.

Após o projeto definido é necessário criar o modelo acoplado que será utilizado no projeto, para tanto deve-se acessar à opção: *File* > *New* > *Other* (demonstrado na figura 25). Após o clique do usuário, uma janela é aberta (figura 26) para escolher o tipo de arquivo para esta abordagem, que será selecionado o modelo acoplado.

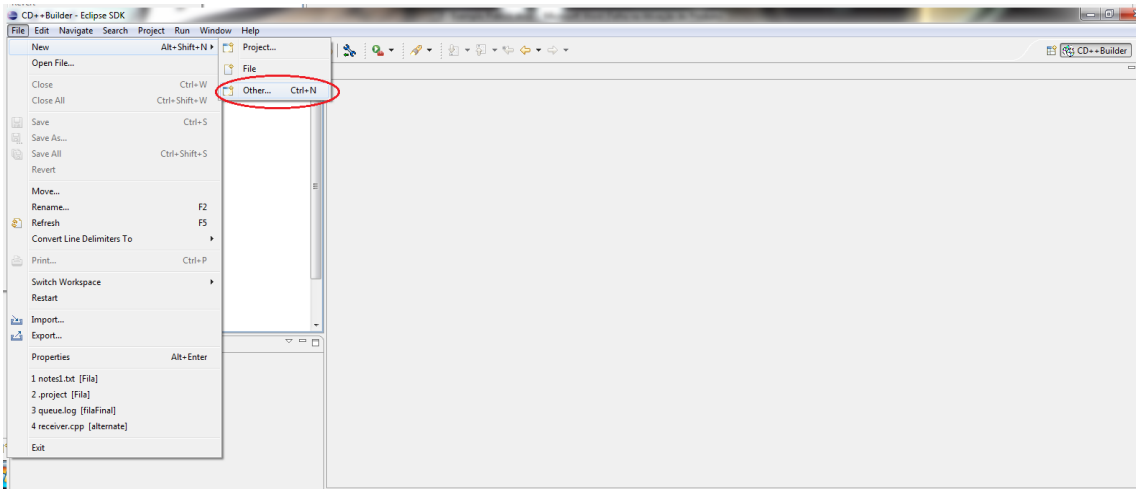

Figura 25: Criando o arquivo de modelagem do projeto (Queue.ma).

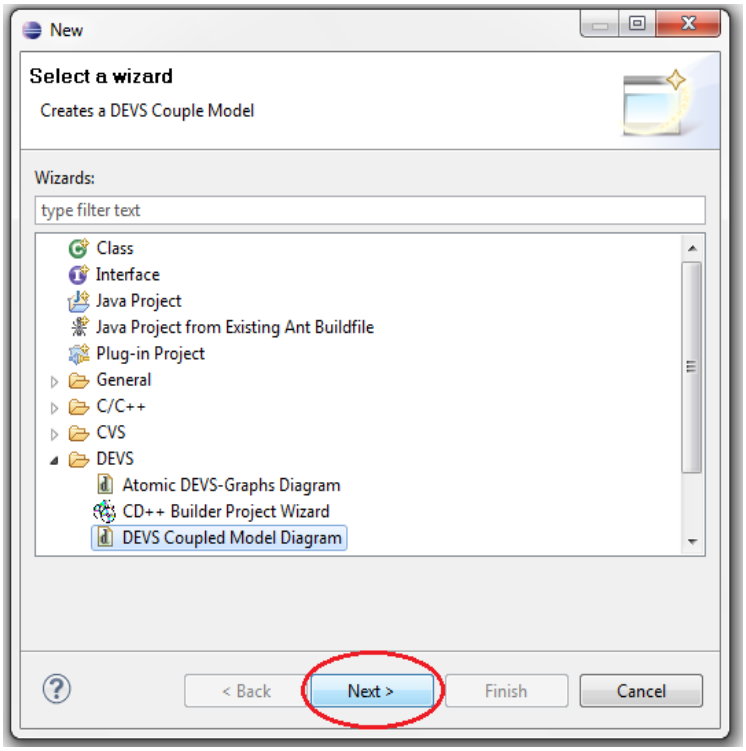

Figura 26: Apresenta o tipo de modelo escolhido, no caso o modelo acoplado.

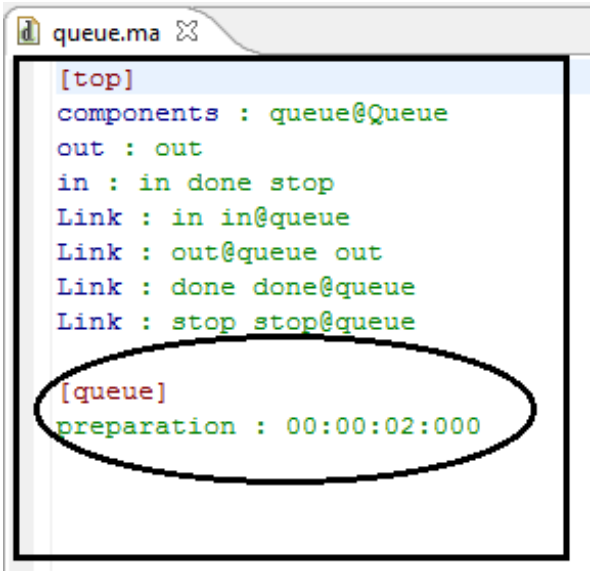

Figura 27: Definição dos componentes no modelo (arquivo Queue.ma).

A figura 27 apresenta a descrição do modelo acoplado da Fila, como já foi abordado anteriormente, todo modelo acoplado (neste caso: Queue.ma) é composto por grupos de definições para os modelos de acoplamento (possibilitando também a configuração de modelos atômicos). A região destacada pelo retângulo apresenta o modelo acoplado e seus grupos de definições. O grupo *[top]* é obrigatório no modelo acoplado, *[Queue]* é outro grupo de definição, mas não obrigatório que representa o modelo atômico, para este projeto há necessidade de configurar quatro propriedades (cláusula *components*, *out*, *in*, *link*).

A cláusula **components** que descreve o(s) modelo(s) que compõe o modelo acoplado, neste caso é composta por um único modelo atômico chamado *queue@Queue (nomeModelo@ClasseDoModelo)*. Quando um componente é declarado, é necessário criar o seu grupo de definição. A figura 27 foi marcada com uma elipse para identificar um grupo de definição, que é o modelo atômico *[Queue]*. Neste modelo é configurado o atributo *preparation*, responsável por indicar o tempo de preparação (*preparationTime*) para os elementos da fila, até que este seja enviado para fora, pela porta de saída.

A cláusula **Out** enumera as portas de saída, para esta modelagem se tem apenas uma porta de saída com o nome *out*.

A cláusula **In** enumera as portas de entrada, para esta modelagem se tem três portas de entrada com o nome *in*, *done* e *stop*.

A cláusula **Link** descreve o esquema interno e externo acoplado (Na Figura 28 há apresentação do modelo em forma de diagrama).

O arquivo de modelagem **queue.ma** pode ser visualizado também em forma de diagrama, a figura 28, apresenta as declarações realizadas na figura 27 para o modelo acoplado Fila em formato de diagrama, para isto é necessário clicar na aba *diagram*.

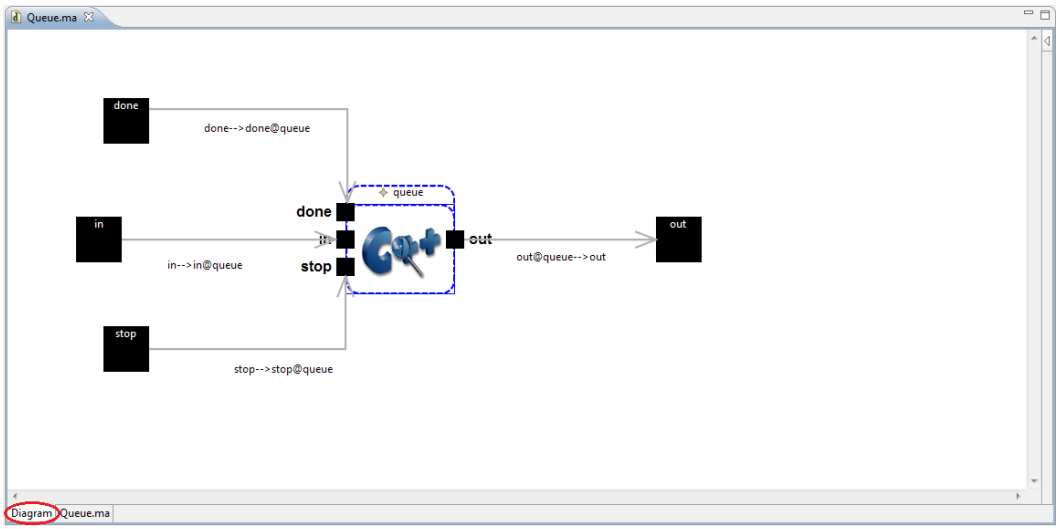

Figura 28: Definição dos componentes do modelo (visão do arquivo Queue.ma em formato de diagrama).

3.2.2 Implementação de um novo modelo atômico

Para a implementação de um novo modelo atômico, se faz necessário definir uma classe derivada de *atomic.h*, sobrecarregando os métodos necessários. A classe *atomic.h* é uma classe abstrata que declara uma interface (Application Program Interface (API)) para criar modelos, e define algumas funções de serviço que o usuário pode usar, redefinindo as classes de base (WAINER, 2009).

No processo de desenvolvimento do modelo atômico da Fila é necessário criar três classes, para tanto deve-se acessar à aba: *File* > *New* > *Other*, e criar os

arquivos: *Register.cpp*, *Queue.h*, *Queue.cpp*. A descrição de como implementar esses arquivos são realizados nas sessões seguintes.

3.2.2.1 Definindo o arquivo Register.cpp

Para definir um novo modelo atômico deve-se criar o arquivo *queue.cpp*, dentro da classe o usuário deve usar o método *registerAtomic*, como é apresentado na figura 29.

```
void MainSimulator::registerNewAtomics()
{
    SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Queue>() , "Queue" ) ;
}
```

Figura 29: Definição do arquivo register.cpp.

Este método registra um novo modelo chamado, para este exemplo de *Queue* que é o nome da classe, e *Queue* também é o nome que será usado para identificar o modelo de fila nos outros arquivos (incluindo o arquivo de modelo acoplado). Desta forma, os usuários podem dar nomes significativos para os seus modelos, mesmo se eles forem usados para outros fins (WAINER, 2009).

3.2.2.2 Definindo a classe Queue.h e Queue.cpp

Para criar este modelo no CD++, deve-se criar a classe *queue.h* (derivada de *atomic.h*) para armazenar o estado da fila. As funções de serviço (exemplo: *externalFunction*) permitem que o modelo possa definir o estado atual e sua duração. A figura 30, apresenta a classe *queue.h* com as definições necessárias para implementar o modelo de fila, incluindo as portas de entrada/saída e variáveis de estado. A lista de valores que contém os dados de entrada (*ElementList*) é

definida usando a biblioteca de modelo padrão e *TimeLeft* é usado para armazenar o tempo restante, se o modelo é interrompido por um sinal de controle de fluxo (WAINER, 2009).

A fila é transmitida através da porta de saída. A porta de entrada *stop* serve para regular o fluxo: Se receber uma mensagem da porta de entrada *stop*, que desative temporariamente a fila (ou seja, ele só responde a novos eventos recebidos através da porta de entrada). Qualquer entrada recebida será armazenada, mas nenhuma saída será enviada até que a fila esteja ativada novamente (enviando outra mensagem para a porta *stop*). O parâmetro *preparationTime* é usado para modelar o atraso do dispositivo de enfileiramento (WAINER, 2009).

```
class Queue : public Atomic
{
public:
    Queue( const string &name = "Queue" ); // Construtor
    virtual string className() const ;

protected:
    Model &initFunction();
    Model &externalFunction( const ExternalMessage & );
    Model &internalFunction( const InternalMessage & );
    Model &outputFunction( const InternalMessage & );

private:
    const Port &in;
    const Port &stop;
    const Port &done;
    Port &out;
    Time preparationTime;
    typedef list<Value> ElementList ;
    ElementList elements ;
    Time timeLeft;

};
```

Figura 30: Definição da interface Queue.h.

A classe *queue.cpp* é responsável por sobrescrever os métodos: de inicialização (*initFunction*), transição interna (*internalFunction*), transição externa (*externalFunction*), e métodos de saída (*outputFunction*) da interface *queue.h*.

A figura 31 demonstra o método construtor do modelo atômico da fila, neste são definidos alguns atributos, como as portas de entrada *in*, *stop*, *done* a porta de saída *out* e o tempo de preparação dos elementos (*preparationTime*).

```
Queue::Queue( const string &name ) :  Atomic( name )
, in( addInputPort( "in" ) )
, stop( addInputPort( "stop" ) )
, done( addInputPort( "done" ) )
, out( addOutputPort( "out" ) )
, preparationTime( 0, 0, 10, 0 )

{
    string time( MainSimulator::Instance().getParameter( description(), "preparation" ) ) ;
    if( time != "" )
        preparationTime = time ;
}
```

Figura 31: Método Construtor

3.2.2.3 Execução dos métodos

Na função de inicialização (*initFunction,* figura 32), as variáveis do modelo têm o valor inicial e todos os valores da fila são eliminados.

```
Model &Queue::initFunction()
{
    elements.erase( elements.begin(), elements.end() ) ;
    return *this ;
}
```

Figura 32: Função de Inicialização.

A função de transição externa (*externalFunction,* figura 33) é responsável por controlar o fluxo de elementos externos nas portas de entrada. Quando um evento externo vem de uma porta de entrada (*in*), o valor é inserido na fila interna e, em seguida, é verificado se o estado da fila permite que seja programado para efetuar uma nova remessa para a porta de saída. Se a mensagem chegou para a porta *Done* o último elemento enviado pode ser eliminado da fila interna e se prepara o próximo (se existir). Se a mensagem vem da porta *Stop* o conteúdo deve ser analisado para interpretar a ordem como "parar" ou "continuar" a expedição de dados. Se ele parar, em seguida, registra o tempo restante para concluir a iteração a ser considerado quando renovar as tarefas (WAINER, 2009).

Um evento externo que chega na porta de entrada (*in*) representa um novo valor de entrada, o valor é inserido na fila interna, se é o único elemento na fila, tem que ser retransmitido imediatamente (WAINER, 2009).

Um evento que chega à porta *done* indica que o último elemento enviado foi processado, tem que ser enviado para fora da fila interna, e um próximo elemento ficaram aguardando (se existir). Havendo mais elementos a serem transmitidos, o primeiro valor na fila deve ser preparado (WAINER, 2009).

Quando um evento chega à porta *stop* indica que o fluxo deve ser interrompido ou reiniciado. Se a fila estava no estado ativo e o valor de mensagem não é zero, a fila é pausada. Neste momento, o tempo restante para processar a mudança de estado seguinte é calculado (fim do tempo de preparação) e, em seguida, a fila altera o seu estado para passivo (*passive*), chamando o método *passive()*. Se a fila estava em estado passivo (*passive*) e o valor da mensagem é zero, então a fila reiniciará, e a próxima mudança de estado está prevista após o tempo de processamento restante (WAINER, 2009).

```
Model &Queue::externalFunction( const ExternalMessage &msg )
{

    if (msg.port() == in ){
        elements.push_back(msg.value());
        if(elements.size() == 1)
            this->holdIn(active, preparationTime);
    }
    if( msg.port() == done )
    {
        elements.pop_front();
        if( !elements.empty() )
            this->holdIn( active, preparationTime );
    }
    if( msg.port() == stop )
        if( this->state() == active && msg.value() )
        {
            timeLeft = msg.time() - this->lastChange();
            this->passivate();
        }
        else
            if( this->state() == passive && !msg.value() )
                this->holdIn( active, timeLeft );
    return *this;
}
```

Figura 33: Função de transição externa.

A função de saída (*outputFunction*, figura 34) indica que o tempo de preparação (*preparationTime*) para o primeiro elemento da fila concluiu, este será enviado para fora pela porta de saída. Em seguida, a função de transição interna (*internalFunction*, figura 35) é executada indicando que ela terminou de enviar o valor, neste momento, não há nada a fazer a não ser esperar seu reconhecimento na porta *done*. Por conseguinte o modelo altera a sua fase para *passive*. O ciclo vai continuar com a próxima mensagem externa (Wainer, 2009).

```
Model &Queue::outputFunction( const InternalMessage &msg )
{
    this->sendOutput( msg.time(), out, elements.front() );
    return *this;
}
```

Figura 34: Função de saída.

```
Model &Queue::internalFunction( const InternalMessage &msg )
{
    this->passivate();
    return *this;
}
```

Figura 35: Função de transição interna.

Após o término da configuração das classes do modelo atômico será necessário compilar o projeto. A figura 36 apresenta o ícone de compilação da ferramenta CD++, quando o usuário acessa seu projeto será compilado e verificado se está pronto para realizar simulações, a figura 37 ilustra a compilação com sucesso do projeto Fila.

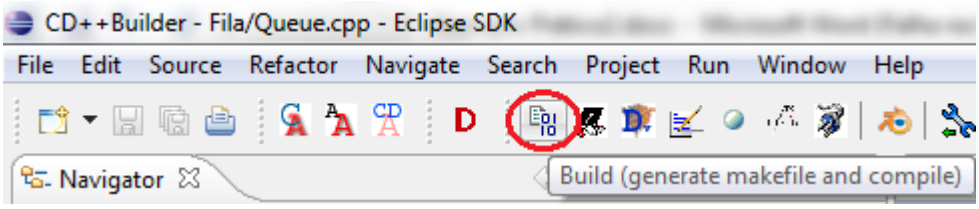

Figura 36: Ícone para compilação do projeto no CD++.

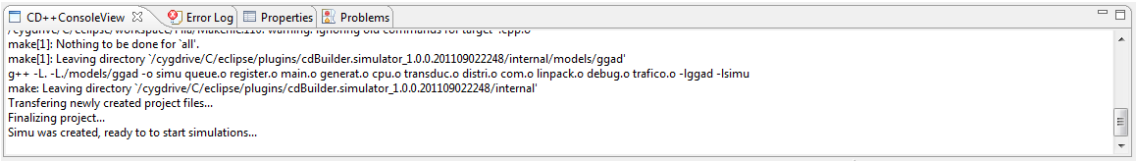

Figura 37: Compilação realizada com sucesso, pronto para realizar simulação.

3.2.2.4 Definição do arquivo de evento externo

A fim de permitir a configuração de parâmetros em tempo de execução, alguns dos argumentos utilizados pelos modelos atômicos podem ser definidos externamente. Para criar o arquivo de evento externo é necessário ir à aba: *File* > *New* > *File* e definir um nome e necessariamente colocar a extensão *.EV*, para este projeto é definido como *queue.EV*. A figura 38 apresenta o arquivo de evento externo *queue.EV* e seus parâmetros configurados, no instante de dez segundos um evento chega à porta de entrada (*in*) com valor 1.5, em seguida com dezoito segundos um evento chega a porta pronto (*done*), em seguida mais quatro eventos ocorrem alternando as portas *in* e *done* e seus valores.

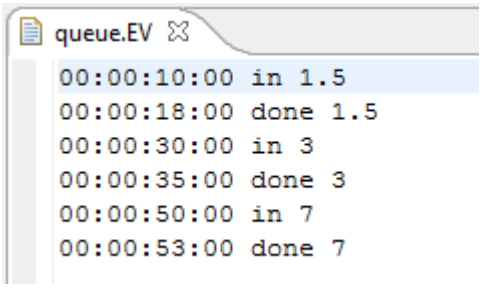

Figura 38: Definição do arquivo de eventos externos.

3.2.2.5 Simulação

Com o projeto pronto e seu arquivo de evento externo criado, neste momento já é possível realizar uma simulação, para isso deve-se clicar no ícone como é apresentado na figura 39. Será necessário colocar o caminho de alguns arquivos do nosso projeto como *queue.MA*, *queue.EV*, *queue.LOG* e *queue.OUT*, os arquivos *queue.LOG* e *queue.OUT* ainda não foram criados, porém serão criados automaticamente no processo de simulação basta declará-los como a figura 40. O tempo de simulação também foi configurado, em um minuto.

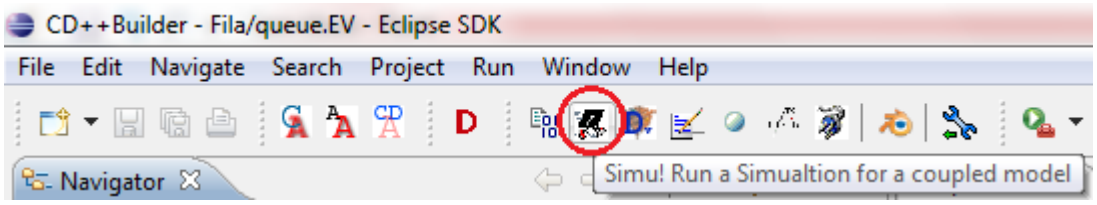

Figura 39: Ícone de simulação do modelo.

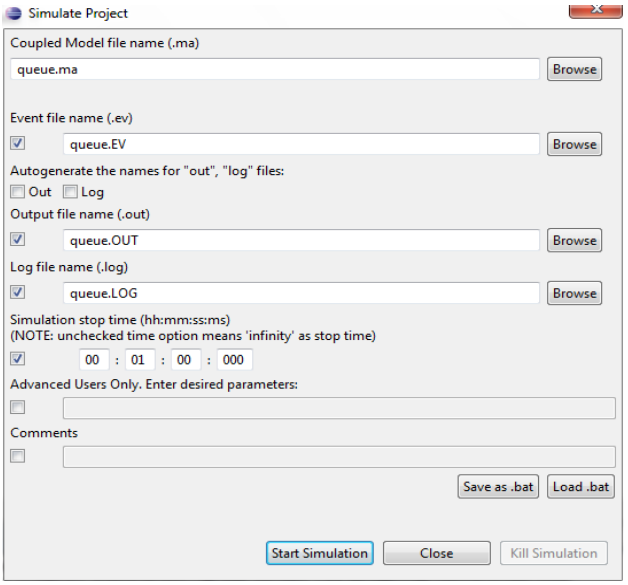

Figura 40: Simulação do projeto.

3.2.2.6 Verificação da simulação

Após a simulação ter sido realizada com sucesso, deve-se verificar o arquivo *queue.OUT,* este gera a saída do modelo de fila. A figura 41 apresenta o tempo em que os elementos chegaram à porta de saída, detalha os instantes: 12, 32 e 52 segundos e seus respectivos valores 1.5, 3 e 7. Já a figura 42 apresenta o arquivo de log gerado pelo modelo de fila.

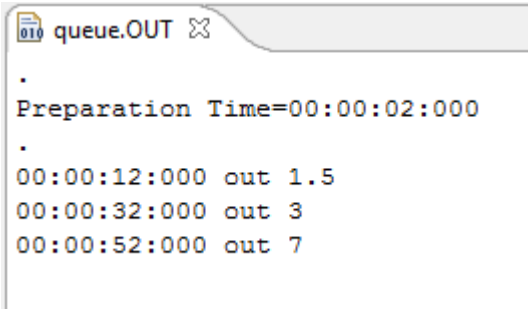

Figura 41: Saída gerada pelo modelo no arquivo queue.OUT.

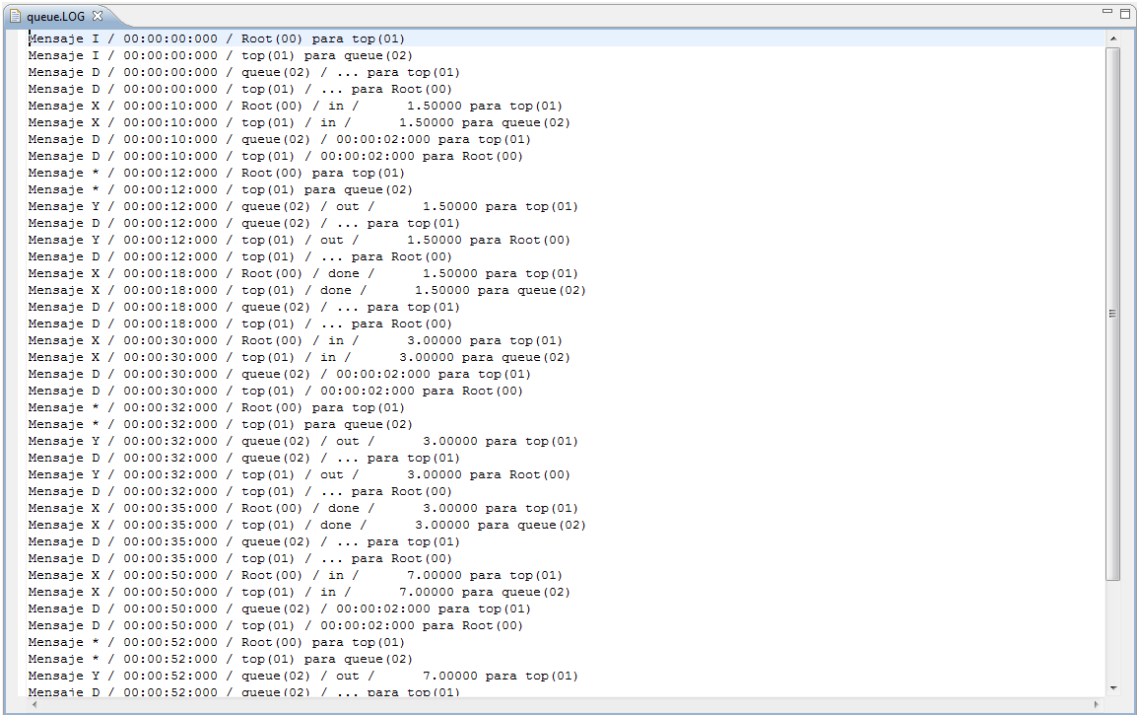

Figura 42: Arquivo de log, queue.LOG.

3.2.3 EXEMPLO DE MODELAGEM CELULAR, JOGO DA VIDA

Neste capitulo será apresentado um exemplo de modelo celular DEVS para o jogo da Vida (Life Game), demonstrando seu funcionamento, para as principais etapas do jogo.

3.2.3.1 Exemplo – Jogo da Vida

Descrição: O "jogo da vida" com as regras originais propostos por Conway (Matemático Inglês, ativo na teoria dos grupos finitos, teoria combinatória dos jogos). A regra fundamental é conhecida como "B3/S23". A nova célula nasce quando ela tem exatamente 3 vizinhos. Uma célula existente sobrevive se tem 2 ou 3 vizinhos. Caso contrário, ela morre (Gardner, 1970).

O modelo célula DEVS é um caso especial do modelo acoplado, onde as definições de um modelo celular são necessárias para modelar as dimensões da célula, tipo de atraso, valores padrões iniciais e regras de transição locais (WAINER, 2009).

3.2.3.2 Modelo Jogo da Vida

No modelo proposto, cada célula pode ser ocupada por uma entidade viva (valor = 1), ou por uma célula vazia (célula morta, valor = 0). Uma célula viva permanece viva apenas se têm três ou quatro vizinhos vivos, caso contrário ela morre. Uma célula se torna viva quando possui exatamente três vizinhos vivos, próximos de uma célula vazia. O modelo de célula DEVS acoplado no exemplo é definido pelo seu tamanho (largura = 20, altura = 20), a sua borda (pacote, que possui células em uma margem que se comunica com os resultados para os vizinhos na margem oposta). As regras definidas no jogo representam o comportamento de cada célula no modelo. Neste caso, uma célula ativa ((0,0) = 1),

permanece ativa quando o número de vizinhos ativo é igual a três ou quatro, utilizando um atraso no transporte de 100ms. Se a célula está inativa ((0,0) = 0) e sua vizinhança possui três células ativas, ela se torna ativa. Em qualquer outro caso, a célula permanece inativa (indica que sempre que a regra é avaliada, um valor verdadeiro é retornado) (WAINER, 2009).

```
[top]
components : life

[life]
type : cell
width : 20
height : 20
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1)  life(0,0)  life(0,1)
neighbors : life(1,-1)  life(1,0)  life(1,1)
initialvalue : 0
initialrowvalue :  5     00000001110000000000
initialrowvalue :  7     00000100100100000000
initialrowvalue :  8     00000101110100000000
initialrowvalue :  9     00000100100100000000
initialrowvalue : 11     00000001110000000000
localtransition : life-rule


[life-rule]
rule : 1 100 { (0,0) = 1 and trueCount = 5 }
rule : 1 100 { (0,0) = 0 and trueCount = 3 }
rule : 0 100 { t }
```

Figura 43 - Apresenta a definição de um modelo célula DEVS para o Jogo da Vida.

A figura 43 especifica com detalhes o modelo celular para o exemplo Jogo da Vida. Nela é definido o nome do projeto, tipo de modelo, no caso celular, as dimensões do modelo, o tipo de atraso, o tempo de atraso e a predisposição dos valores na célula, valor inicial padrão (*default*) das células, os valores inicializados em cada linha, e por fim, a lógica do modelo que mostra a condição para uma determinada célula viver, ou seja, ter o valor um ou permanecer morta, ter valor zero.

3.2.3.3 Carregando o modelo através da IDE Eclipse, utilizando o plugin CD++.

Nesta etapa deve ser criado um projeto no Eclipse do tipo *CD++ Builder Project Wizard*. Após criado, selecionar com o botão direito do mouse a opção *import* e depois o item *Archive File* e na próxima janela o arquivo zip, rar, etc, que está salvo no computador, deve ser selecionado. Se carregado com sucesso o projeto será mostrado no lado esquerdo da tela, como mostrado na figura 44.

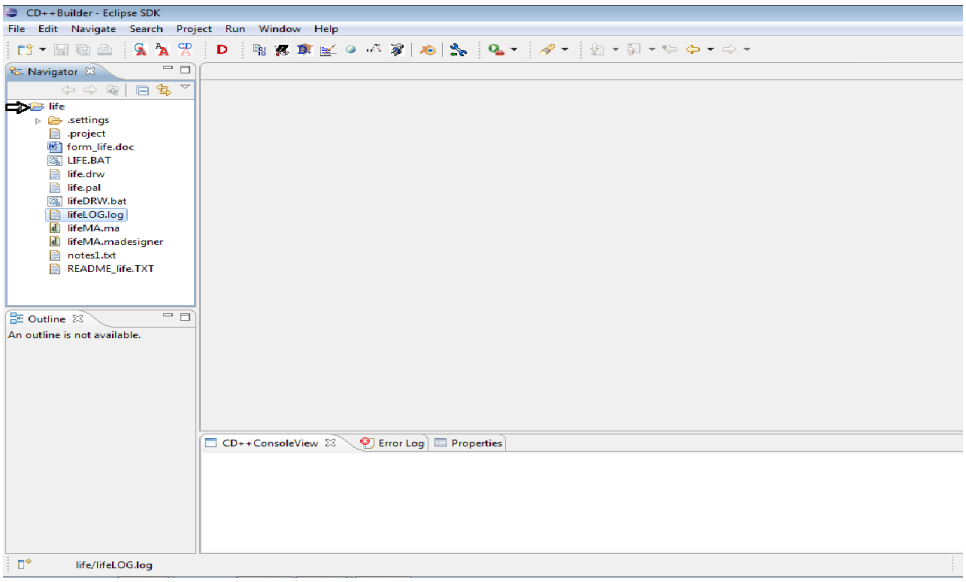

Figura 44 - Jogo da Vida carregado no Eclipse CD++.

3.2.3.4 Compilando o projeto criado

Após carregado, o modelo deve ser compilado, com a finalidade de realizar a simulação dos valores de entrada. A opção *Animate Cell-Dev Simulation* deve ser selecionada, para que uma nova janela se abra, onde os dados para simulação serão carregados. A figura 45 demonstra essa etapa.

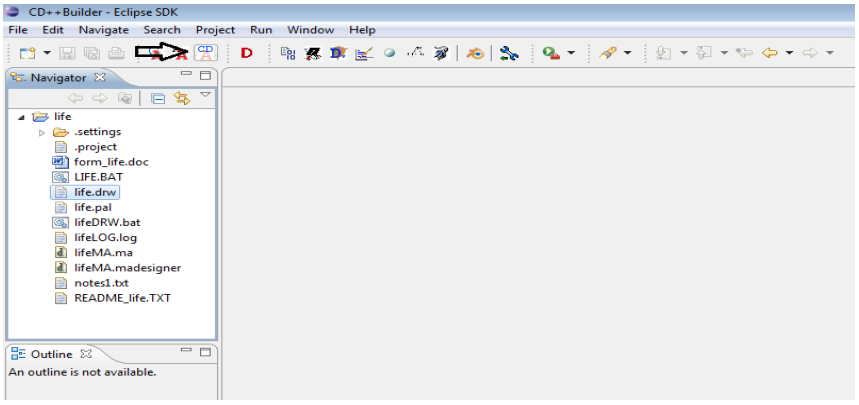

Figura 45 - Opção "AnimateCell-DevSimulation" selecionada.

Na próxima janela, o botão *ADD Model* deve ser selecionado para que o arquivo *life.drw* possa ser carregado, com os dados da simulação. A figura 46 demonstra a janela, com os campos que devem ser selecionados.

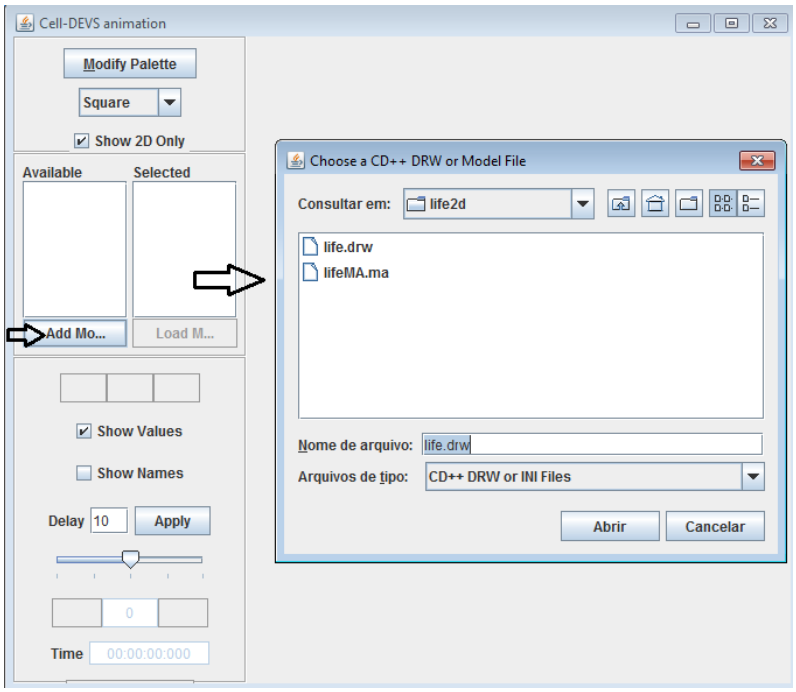

Figura 46 - Arquivo "life.drw" adicionado para simulação.

Após ser adicionado o arquivo no campo *Available*, o mesmo deve ser selecionado para opção *Selected*. Para isso deve ser pressionado o botão esquerdo

do mouse duas vezes. No próximo passo deve ser selecionado o botão *Load Model* para que o modelo seja carregado na tela.

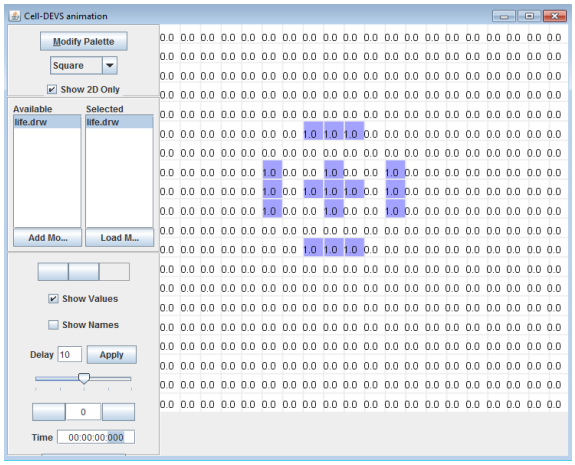

Figura 47 - Modelo “life” carregado no formato de Células.

Com o arquivo carregado, é possível realizar algumas modificações, como: alterar a fonte, cor do modelo, selecionando a opção *Modify Palette, Delay* (modificar o tempo de exibição dos dados), alterar a forma de exibição das células e por fim selecionar o intervalo de tempo para execução da simulação (WAINER, 2009).

Para executar a simulação se faz necessário selecionar um dos botões que estão acima da opção *Show Values*. Depois de realizada a simulação, um arquivo de log, *lifeLOG.log* será gerado para análise; esse arquivo contém as diferentes situações do modelo no transcorrer do tempo (WAINER, 2009).

A figura 48 apresenta com detalhe arquivo de Log do modelo Jogo da Vida, onde são divididos por instantes de tempo, começando em 000 até 300 milissegundos.

```
Time: 00:00:00:000
Root(00) para top(01)
top(01) para life(02)
life(02) para life(0,0)(03)
life(02) para life(0,1)(04)
life(02) para life(0,2)(05)
life(02) para life(0,3)(06)
life(02) para life(0,4)(07)
life(02) para life(0,5)(08)
life(02) para life(0,6)(09)
Time: 00:00:00:100
Root(00) para top(01)
top(01) para life(02)
life(02) para life(4,8)(91)
life(02) para life(5,7)(110)
life(02) para life(6,7)(130)
life(02) para life(6,9)(132)
life(02) para life(7,5)(148)
Time: 00:00:00:200
Root(00) para top(01)
top(01) para life(02)
life(02) para life(4,8)(91)
life(02) para life(6,9)(132)
life(02) para life(6,10)(133)
life(02) para life(7,6)(149)
Time: 00:00:00:300
Root(00) para top(01)
top(01) para life(02)
life(02) para life(5,8)(111)
life(02) para life(6,6)(129)
life(02) para life(6,10)(133)
life(02) para life(8,11)(174)
life(02) para life(10,6)(209)
```

Figura 48: Log do modelo Jogo da Vida

3.2.3.5 Análise da produção e validação

A partir do estado inicial as células ficam dispostas, de modo que as células com valor igual a um, que representam a entidade viva ficam concentradas no centro do modelo, ficando em destaque com relação às outras células, mortas, com valor igual a zero (WAINER, 2009).

A saída do modelo mostra o processo de validação das regras definidas, onde são executadas as restrições para o modelo. Uma célula está viva se três ou quatro de seus vizinhos tem o mesmo status, caso contrário ela morre. Para que a célula altere seu status para viva, é necessário ter três vizinhos vivos, próximos dessa célula que estava morta (WAINER, 2009).

No Tempo: 00:00:00:300, a entrada para life onde as células são (5,8), indicam que seus valores devem ser verificados nas condições da lógica

determinada, para atribuir se esses registros podem ser vivos ou ficarem como mortos (WAINER, 2009).

A partir do resultado acima, podemos encontrar a interação de sucesso do modelo com as regras definidas. As regras que determinam cada ação de uma determinada célula representam a interação dinâmica do sistema (WAINER, 2009).

3.2.4 EXEMPLO DE MODELAGEM ACOPLADA E ATÔMICA, PROTOCOLO BIT ALTERNADO

Uma das principais áreas de aplicação da modelagem discreta de evento e simulação é a análise de protocolos de rede. A atual escala dessas redes e seu elevado nível de heterogeneidade tornam muito difíceis o detalhamento de novos projetos envolvendo diferentes protocolos. Diversos simuladores estão prontamente disponíveis (tanto acadêmico quanto comercial, como a NS-2 e seu sucessor NS-3, GloMoSim, OPNET, e OMNeT + + (Wainer, 2009).

A utilização de um simulador de rede baseado em DEVS oferece facilidades no que diz respeito à implementação de testes formais, compartilhamento entre modelos diferentes DEVS, bem como a possibilidade de definir modelos usando diferentes técnicas, no mesmo ambiente. Isso pode levar inclusive fora da rede entidades que afetam a operação de rede, fornecendo resultados que são mais realistas (WAINER, 2009).

ABP (Protocolo Bit Alternado) é um protocolo de comunicação para garantir a transmissão confiável através de uma rede não confiável, onde o remetente envia um pacote e espera por uma confirmação. Se a confirmação não chegar dentro de um tempo pré-definido, o remetente reenvia este pacote até que receba um reconhecimento esperado e, em seguida, envia o pacote seguinte. A fim de distinguir dois pacotes consecutivos, o emissor adiciona um bit adicional em cada pacote (chamado bit alternado porque o remetente utiliza o número 0 e 1 como alternativa). Um modelo DEVS chamado Simulador ABP é criado para simular o comportamento do protocolo Bit alternado. O Simulador ABP consiste em três componentes: rede, o remetente e o receptor. A rede é decomposta de mais duas sub-redes correspondentes ao envio e recebimento de canal respectivamente, a figura 49 detalha a estrutura de interação do modelo de Protocolo Bit Alternado (WAINER, 2009).

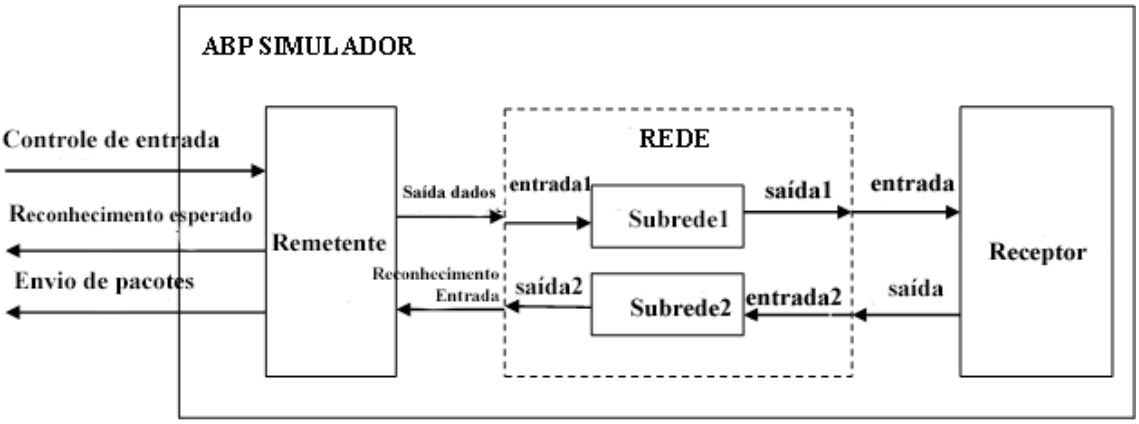

Figura 49: Estrutura do modelo de protocolo Bit Alternado.

O comportamento do receptor consiste em receber os dados e enviá-los de volta com um reconhecimento extraído a partir dos dados recebidos, após um período de tempo. Nas sub-redes os dados são recebidos após um atraso de tempo. No entanto, a fim de simular a falta de segurança (confiabilidade) da rede, apenas 95% dos dados serão transmitidos em cada sub-rede, ou seja, 5% dos dados serão perdidos através da sub-rede (WAINER, 2009)

O receptor e as sub-redes possuem duas fases: ativa e passiva. Eles estão na fase passiva inicialmente. Sempre que receber uma entrada, eles vão estar em fase ativa, e enviar uma saída (com uma probabilidade de 95% na sub-rede) depois de um tempo de duração. O estado será alterado novamente para a fase passiva. O *receiveing_time* do receptor é uma constante, enquanto o atraso nas sub-redes é não determinístico e seu valor é expresso por uma distribuição normal com média e desvio (WAINER, 2009).

O comportamento do remetente é muito mais complicado. Seu estado depende das seguintes variáveis de estado definidas pelo usuário: *Alt_bit*, *sending*, *ACK*, e *packetNum*. O remetente realiza as mudanças da fase inicial passiva para ativa quando um sinal *controlIn* é recebido. Após esse sinal um modo de envio, para encaminhamento com um pacote de mais um bit alternado. Quando um *sending_time* é decorrido, o pacote é assumido para ser enviado para um destinatário, e o remetente permanece aguardando o seu reconhecimento. Se o tempo limite expirar, o remetente irá reenviar o pacote. Se o reconhecimento esperado é recebido antes do tempo limite, o remetente envia o próximo pacote. Ele vai mudar novamente para a fase passiva quando todos os pacotes forem enviados

com sucesso. A saída será gerada quando um pacote é enviado para fora (*packeSent*, *DATAOUT*) ou um reconhecimento esperado é recebido (*ackReceived*). Para simplificar, o pacote que foi enviado pelo remetente apenas é considerado o número de sequência do pacote mais um bit alternado (por exemplo, 11 para o primeiro pacote, 100 para o décimo pacote). Assim, o número de sequência do pacote (por exemplo, 1 para o primeiro pacote, 10 para o décimo pacote), é enviado para a porta *packetSent* enquanto que o número de sequência do pacote mais o bit alternado (por exemplo, 11 para o primeiro pacote, 100 para o décimo pacote 10) são enviados para a porta de *DATAOUT*. O sinal *controlIn* é um inteiro positivo que indica quantos pacotes devem ser enviados em uma sessão (WAINER, 2009)

Como mostrado na figura 49, o protocolo possui uma entrada e duas saídas. A entrada *controlIn* indica o número de pacotes que devem ser enviados. As duas saídas mostram o tempo quando um pacote foi enviado com sucesso (*packetSent*) ou um reconhecimento esperado é recebido (*ackReceived*). O Simulador ABP consiste em três componentes: rede, remetente e receptor. O remetente envia os pacotes para o receptor e recebe o reconhecimento a partir do receptor por meio da rede. A rede pode ser adicionalmente decompostas em dois componentes de sub-rede, que indicam dois canais. As duas sub-redes são assumidas para ter o mesmo comportamento e utilizar a mesma classe de sub-rede (WAINER, 2009). A figura 50 apresenta o código estruturado do modelo atômico do Protocolo Bit Alternado, onde os principais métodos envolvidos na interação do modelo são descritos.

```
Sender::Sender( const string &name ): Atomic( name ), controlIn( addInputPort( "controlIn") )
  , ackIn( addInputPort( "ackIn" ) ), dataOut( addOutputPort( "dataOut" ) )
  , packetSent( addOutputPort( "packetSent" ) ), ackReceived( addOutputPort( "ackReceived") )
  , preparationTime( 0, 0, 10, 0 ), timeout(0, 0, 20, 0){

   alt_bit = 0;
}

Model &Sender::externalFunction( const ExternalMessage &msg ) {
   if( msg.port() == controlIn  && state() == passive)       {
        totalPacketNum = static_cast < int > (msg.value());
        if (totalPacketNum > 0)       {
                packetNum = 1;
                ack = false;
                sending = true;
                alt_bit = packetNum % 2;       //set initial alt_bit
                holdIn(active, preparationTime );
        }
   }
   if( msg.port() == ackIn  && state() == active)    {
        if (alt_bit == static_cast < int > (msg.value()))           {
                ack = true;
                sending = false;
                holdIn(active, Time::Zero) ;
        }
   }
}


   Model &Sender::internalFunction( const InternalMessage & ){
   if (ack)       {
      if (packetNum < totalPacketNum)           {
           packetNum ++;
           ack = false;
           alt_bit = (alt_bit + 1) % 2;
           sending = true;
           holdIn( active, preparationTime );
           }
      else
           passivate();
      }
   else    {
      if (sending)        {
           sending = false;
           holdIn(active, timeout);
           }
      else         {
           sending = true;
           holdIn( active, preparationTime );
           }
      }
   }

   Model &Sender::outputFunction( const InternalMessage &msg ){
   if (sending)    {
      sendOutput( msg.time(), dataOut, packetNum * 10 + alt_bit ) ;
           sendOutput(msg.time(), packetSent, packetNum ) ;
      }
      else
           if (ack)  sendOutput( msg.time(), ackReceived, alt_bit ) ;
   }
```

Figura 50: Implementação do modelo atômico do Protocolo Bit Alternado

3.2.4.1 Análise de Execução

A entrada do receptor é passada como um número inteiro positivo. O último dígito do número inteiro deverá ser o número zero ou um, que indica o bit alternado. Outros dígitos são assumidos como sendo os dados do pacote. A saída do receptor (reconhecimento) é o bit alternado extraído a partir da entrada. Na saída são gerados após um período de tempo fixo (por exemplo, 10 unidades de tempo) quando a entrada é recebida. O receptor só deve trabalhar com um pacote de cada

vez. Se um novo pacote chega ao passo que o receptor está processando um pacote, o pacote mais velho deve ser descartado. Nos casos de teste, se o tempo de duração entre duas entradas consecutivas for igual ou menor do que o *receiving_time* do receptor, a antiga entrada deve ser descartada, e nenhuma saída será gerada para essa entrada. A figura 51 apresenta o arquivo de evento *receiver.ev*, com os instantes de duração de determinados eventos de entrada. Ele contém eventos normais e eventos consecutivos com tempo de duração inferior ou igual à *receiving_time* (ou seja, 10 unidades de tempo) do receptor (00:00:10:00 in 11) (WAINER, 2009).

```
00:00:30:00 in 20
00:00:45:00 in 31
00:00:52:00 in 31
00:01:25:00 in 40
00:01:35:00 in 40
00:01:55:00 in 51
```

Figura 51: Arquivo de evento de entrada, receiver.ev

Os dois eventos com fontes em **negrito** apresentados na figura 51, não devem gerar quaisquer saídas, pois o próximo evento vem muito rápido para fazê-lo ser descartado. As saídas de outros eventos devem ser os últimos dígitos dos números inteiros de entrada depois de dez unidades de tempo. O arquivo *receiver.out* descrito na figura 52, apresenta as saídas e os resultados esperados para determinado período de tempo (executar *receiver.scp*) (WAINER, 2009).

```
00:00:20:000 out 1
00:00:40:000 out 0
00:01:02:000 out 1
00:01:45:000 out 0
00:02:05:000 out 1
```

Figura 52: Arquivo de saída, receiver.out

No arquivo de saída *sender.out* (figura 53) é apresentado a saída dos eventos. Os eventos com fontes em **negrito** e *itálico* são eventos ilegais que devem ser ignorados. Os dois eventos com fontes em **negrito** são eventos normais, que poderiam causar o remetente para reenviar o pacote anterior. Um evento **(00:01:30:00 ackIn 0)** simula um pacote perdido, o outro evento **(00:02:20:00 ackIn 1)** simula um reconhecimento errado. O simulador funciona como esperado (WAINER, 2009).

00:00:25:000 dataout 11
00:00:25:000 packetsent 1
00:00:30:000 ackreceived 1
00:00:40:000 dataout 20
00:00:40:000 packetsent 2
00:01:10:000 dataout 20
00:01:10:000 packetsent 2
00:01:30:000 ackreceived 0
00:01:40:000 dataout 31
00:01:40:000 packetsent 3
00:01:55:000 ackreceived 1
00:02:05:000 dataout 40
00:02:05:000 packetsent 4
00:02:35:000 dataout 40
00:02:35:000 packetsent 4
00:02:45:000 ackreceived 0
00:02:55:000 ackreceived 1

Figura 53: Arquivo de saída, sender.out

A rede de modelo acoplado consiste de dois modelos de sub-rede. As duas sub-redes funcionam de forma independente. O resultado do teste é semelhante à do submodelo atômico. A figura 54 apresenta parte do arquivo de entrada do arquivo *network.ev* (WAINER, 2009).

00:00:10:00 in1 11
00:00:15:00 in2 1
00:00:20:00 in1 20
00:00:25:00 in2 0
00:00:30:00 in1 31
00:00:35:00 in2 1
00:00:40:00 in1 40
00:00:45:00 in2 0
00:00:50:00 in1 51
00:00:55:00 in2 1
00:01:10:00 in1 60
00:01:15:00 in2 0
00:01:20:00 in1 71
00:01:25:00 in2 1
00:01:30:00 in1 80
00:01:35:00 in2 0
00:01:40:00 in1 91
00:01:45:00 in2 1
00:01:50:00 in1 100
00:01:55:00 in2 0

Figura 54: Arquivo de evento externo network.ev

Devido à função aleatória nos modelos de sub-rede, a saída não é determinista. Várias entradas podem ser perdidas na rede. A figura 55 é parte do arquivo de saída *network.out* (executar *network.scp*), onde dois eventos com fontes em **negrito** são perdidos neste teste (WAINER, 2009).

00:00:12:987 out1 11
00:00:16:796 out2 1
00:00:21:957 out1 20
00:00:28:035 out2 0
00:00:32:182 out1 31
00:00:38:160 out2 1
00:00:48:655 out2 0
00:00:53:446 out1 51
00:01:12:400 out1 60
00:01:18:687 out2 0
00:01:21:055 out1 71
00:01:28:678 out2 1
00:01:33:679 out1 80
00:01:39:759 out2 0
00:01:43:084 out1 91
00:01:47:533 out2 1
00:01:51:704 out1 100
00:01:57:419 out2 0

Figura 55: Arquivo de saída, network.out

O modelo de Simulador ABP simula o protocolo Bit alternado e gera os resultados esperados. Os comportamentos e as características do remetente, receptor e da rede são simulados pelos respectivos modelos. Os dados do pacote são simplificados como um número de sequência de pacote neste modelo. O Simulador ABP funciona exatamente como se esperava de acordo com as especificações (WAINER, 2009).

4 Ferramenta de Modelagem e Simulação Arena

Este capítulo apresenta as principais características da ferramenta para modelagem e simulação de sistemas ARENA, onde seus principais pontos serão abordados.

4.1 Software Arena visão geral

O ARENA é um ambiente gráfico integrado de simulação, que contém todos os recursos para modelagem de processos, desenho e animação, análise estatística e análise de resultados (PARAGON, 2008). O Software apresenta recursos de uma linguagem de simulação e um ambiente de trabalho e experimentação, que pode ser utilizado para testar o modelo e realizar a apresentação de seus resultados.

O software ARENA oferece componentes necessários para modelagem de simulação sem a necessidade de escrever linhas de código ou usar linguagem de programação, sendo que todo processo do desenvolvimento de modelos de simulação é baseado em fluxogramas, de maneira visual e interativa (PARAGON, 2008).

O foco principal da ferramenta é a criação templates, ou seja, uma coleção de objetos/ferramentas de modelagem, que possibilita ao usuário, detalhar o comportamento do processo em análise, através de respostas às perguntas pré-elaboradas, sem programação, de maneira visual e interativa (PARAGON, 2008).

Principais funcionalidades do Software (PARAGON, 2008):

- Modelagem por Fluxogramas;
- Compatível com MS Office e Windows 7;
- Wizard/Assistente: Ajuda na criação de modelos;
- Bibliotecas: Extensa biblioteca de desenhos para interface animada;
- Fiding erros: Facilita o encontro de erros no modelo.

4.1.1 Modelagem no ambiente Arena

A modelagem é desenvolvida a partir de uma visão dos processos do sistema e de suas interações. Pode ser simbolizado tudo que acontece em determinada situação do sistema observado, através de três processos: a criação ou chegada, o serviço ou processamento que é realizado ou recebido de um recurso e a partida (FILHO, 2008).

Estes três processos são representados de forma gráfica no ARENA, que podem ser acessados junto aos painéis ou templates de modelagem no software.

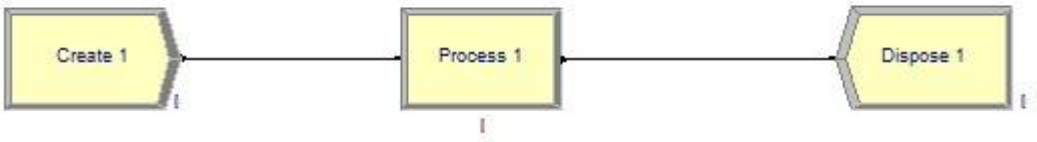

Figura 56: Modelagem básica no ARENA (FILHO, 2008).

A figura 56, apresenta os três módulos presentes no *Template Basic Process*, que permite a implementação da lógica presente nesse sistema. Para acrescentar cada um deles ao modelo desenvolvido, é necessário clicar, arrastar e soltar cada módulo desejado sobre a área de trabalho. Para remover um módulo, basta selecionar e pressionar a tecla delete (FILHO, 2008).

Os três módulos disponíveis no modelo são: **Create** (cria entidades no modelo), **Process** (processos ou serviços executados pelo servidor à entidade) e **Dispose** (retirada das entidades, serviços já executados no sistema). Uma conexão é necessária para relacionar os módulos, ela pode ser obtida através da opção Connect presente na Barra de Ferramentas (FILHO, 2008).

Para fornecer dados do sistema ao modelo de forma representativa para análise é necessário adicionar informações no modulo, para isso basta dar um duplo clique sobre a figura do módulo para que uma caixa diálogo apareça para edição. A

figura 57 apresenta a janela de edição das informações para o módulo **Create**, com campos já preenchidos de valores default (FILHO, 2008).

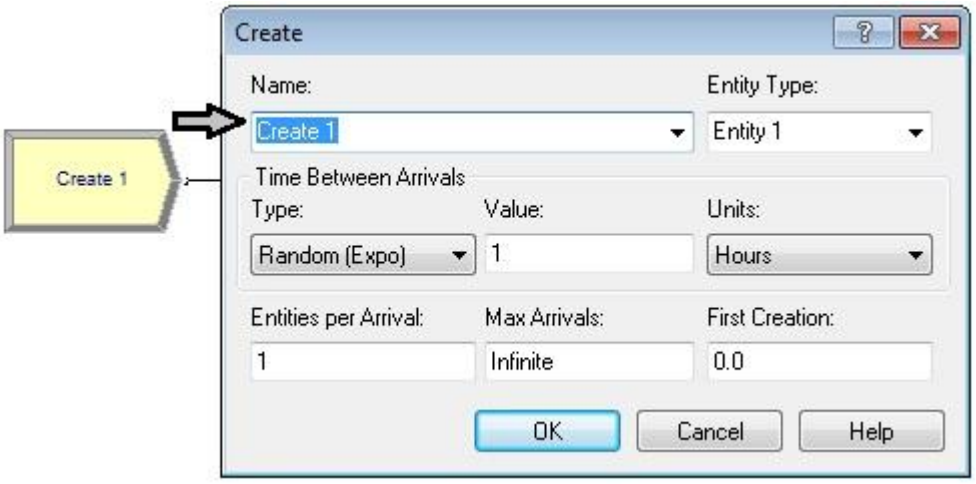

Figura 57: Janela de diálogo padrão do módulo Create.

Os dados fornecidos para o módulo **Create** são utilizados para controlar os processos de chegadas. O primeiro campo *Name*, serve para atribuir um nome ao módulo. O segundo campo, *Entity Type* especifica o nome de um atributo, onde cada entidade criada pelo módulo terá um atributo especificando seu tipo, esses nomes podem ser alterados pelo usuário. Os outros campos da janela servem para controlar as chegadas das entidades, onde os três primeiros com o título *Time Between Arrivals*, especificam o processo de chegadas, seus parâmetros e a unidade de tempo a ser adotada. Os três campos restantes são: determinar quantas entidades serão criadas nas chegadas (padrão 1), restrição quanto ao número de entidades criadas (*Infinite* , sem restrição) e quando ocorrerá a primeira criação (FILHO, 2008).

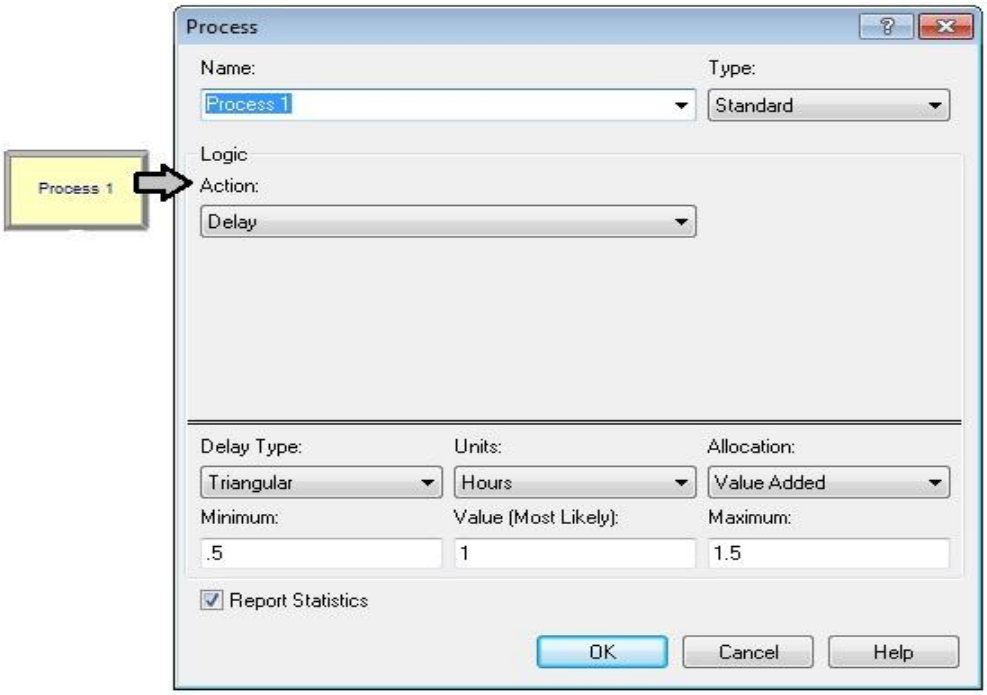

Figura 58: Janela de diálogo padrão do módulo Process.

A figura 58 apresenta a edição dos atributos do módulo **Process**, o procedimento para abrir é o mesmo do módulo **Create**. O primeiro campo *Name*, serve para atribuir um nome ao módulo, no campo *Type* deve ser selecionada a opção *Standard* ou *Submodel* (caso tenha um sub modelo para o **Process**). Na área de Logica a opção *Action* apresenta como opção default o *Delay*, essa opção serve para ativar um atraso para entidade que estiver passando pelo módulo, realizando uma parada ou espera. Caso selecione outra opção o campo *Action* será apresentado com uma sequência de ações (*Seize Delay Release*) que indicam que um recurso será tomado pela entidade (*Size*), essa entidade permanecerá com a posse desse recurso por um determinado tempo (*Delay*) e quando esse tempo de processamento terminar a entidade liberará o recurso (Release) e encaminhara para o próximo módulo. Outros campos serão apresentados quando selecionada a opção (*Seize Delay Release*), como recurso que será utilizado nos processos e outro com as prioridades sobre os recursos. Essas opções podem ser utilizadas em uma fila de prioridade para acessar um determinado recurso. Os últimos campos são relativos ao tempo de processamento (FILHO, 2008).

Figura 59: Janela de diálogo padrão do módulo Dispose.

A figura 59 apresenta o módulo **Dispose**, o procedimento para abrir é o mesmo para os dois últimos módulos apresentados. Esse módulo possui um campo *Name* para seu nome e uma opção *Record Entity Statistics* que quando marcado faz o simulador realizar algumas estatísticas básicas quando executado (FILHO, 2008).

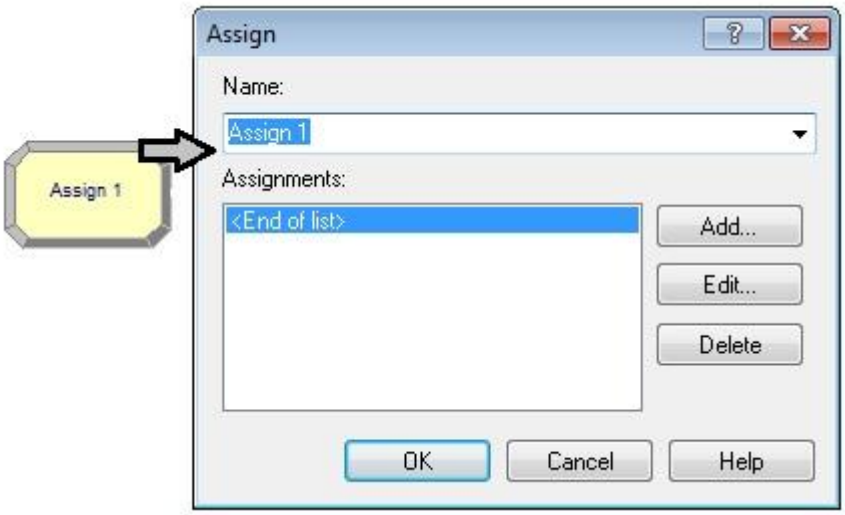

Figura 60: Janela de diálogo padrão do módulo Assign.

A figura 60 ilustra o módulo **Assign**, esse módulo é utilizado para realizar processos de atribuição seguindo uma regra (Variável ou Atributo = Expressão). O

módulo possui um campo *Name* para atribuição do nome e as opções de *Add* (Adicionar tipo do atributo e valor), Edit, Delete (FILHO, 2008).

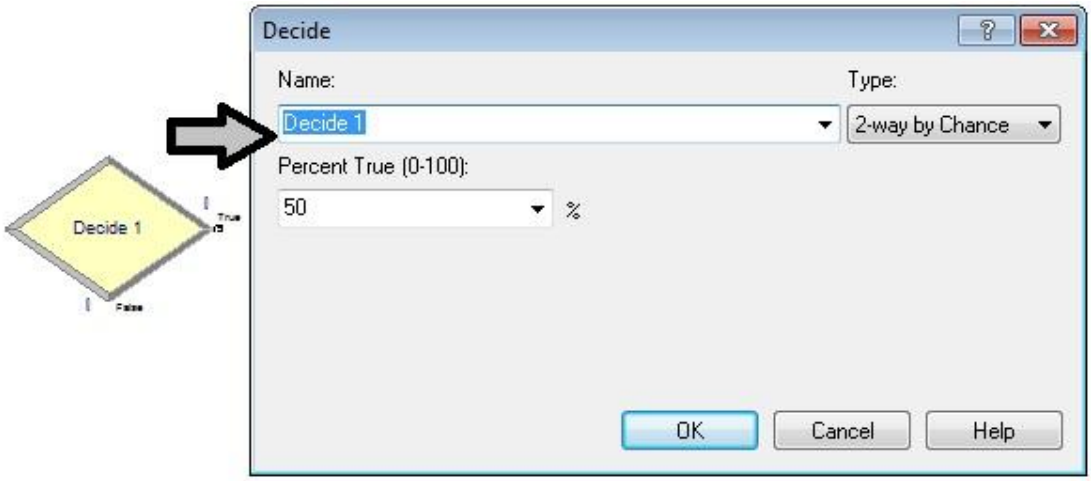

Figura 61: Janela de diálogo padrão do módulo Decide.

A figura 61 apresenta o módulo **Decide,** este pode ser usado em uma modelagem, para estabelecer uma condição e indicar um caminho a ser seguido quando sua condição for Verdadeira ou Falsa. O módulo possui um campo *Name* para atribuição do nome, um campo *Type* que deve ser selecionado para modificar as alternativas de decisão dos processos o padrão utilizado é *2-way by Chance*, que estabelece o número de chances que uma entidade possui para seguir o caminho (*True*) e o campo *Percent True (0-100)*, deve ser selecionado o percentual de chance para uma entidade ser verdadeira (FILHO, 2008).

4.1.2 Exemplo de uma Fila de prioridade

Um sistema de fila pode ter arquiteturas variadas, além de possuir diferentes características de desempenho, dependendo da natureza dos processos de chegada e dos processos de serviços e da maneira imposta pelo seu gerenciamento (FIFO, LIFO, etc) (FILHO, 2008).

O modelo de fila de prioridade proposto, demonstrado na figura 62, possui dois tipos de entidades uma é o cliente comum e outra é o cliente idoso. O objetivo do modelo é dar prioridade ao cliente idoso, ou seja, quando um idoso (cliente especial) chegar para o atendimento no caixa, e caso tenha clientes comuns na fila de espera, o idoso terá prioridade no atendimento e a fila será reorganizada pela ordem de prioridade.

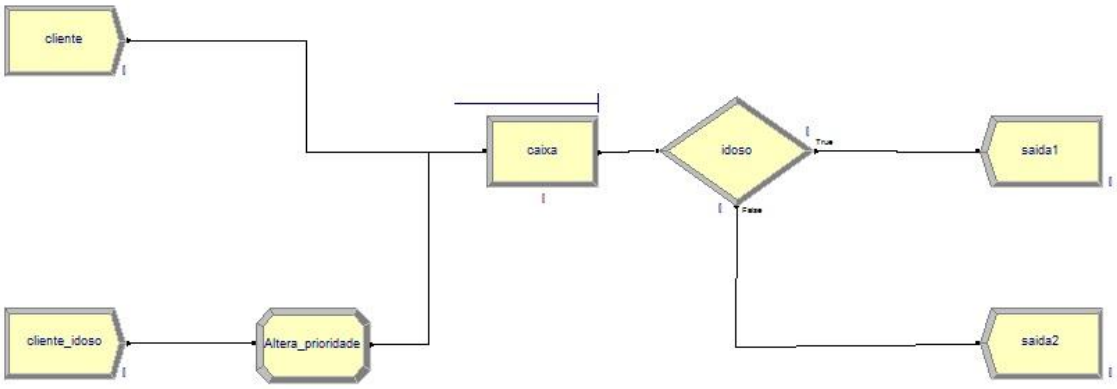

Figura 62: Modelo de uma fila de prioridade no supermercado.

O modelo de fila de prioridade foi criado utilizando sete módulos, a figura 62 apresenta a visão geral do modelo. Neste modelo foram criados dois módulos **Create***: cliente* e *cliente_idoso*, um **Assign** chamado *altera_prioridade*, um **Process** chamado *caixa*, um módulo **Decide** e mais dois **Dispose**, chamados *saída1* e *saída2*. Uma entidade quando sai do módulo *cliente_idoso* passa pelo módulo *altera_prioridade*, neste momento seu atributo de prioridade é alterada para um valor acima de um cliente normal, com isto, a entidade continuará seu trajeto até chegar ao módulo de caixa, neste momento há uma fila de prioridade, que proporciona aos clientes idosos passar na frente dos clientes comuns, reorganizando a ordem da fila. Depois que uma entidade sai do caixa ela passa pelo modelo de decisão, se for um cliente idoso este saíra pela *saida1*, caso contrário *saída2*, esta regra foi adicionada para verificar se a ordem da fila realmente estava sendo obedecida. A seguir as

configurações dos módulos do modelo serão apresentadas e explicadas de forma detalhadas.

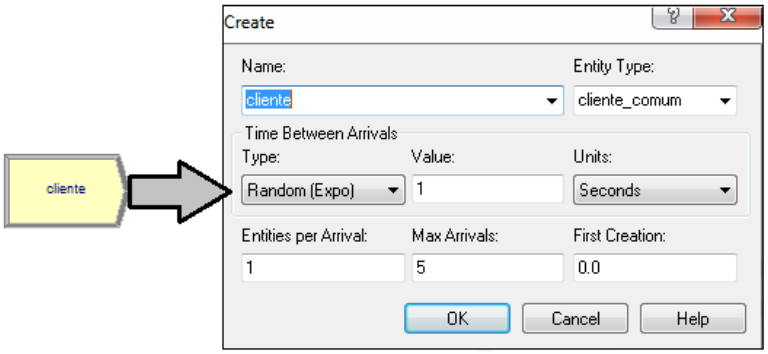

Figura 63: Create do cliente.

A figura 63 apresenta o módulo **Create** referente ao cliente. Nesta figura, são definidos todos os valores dos campos, como o nome do **Create** *cliente_idoso,* tipo de entidade *cliente_especial*. O tempo entre a chegada das entidades também é definido em um segundo, assim como o número máximo de entidades que chegarão que corresponde a cinco.

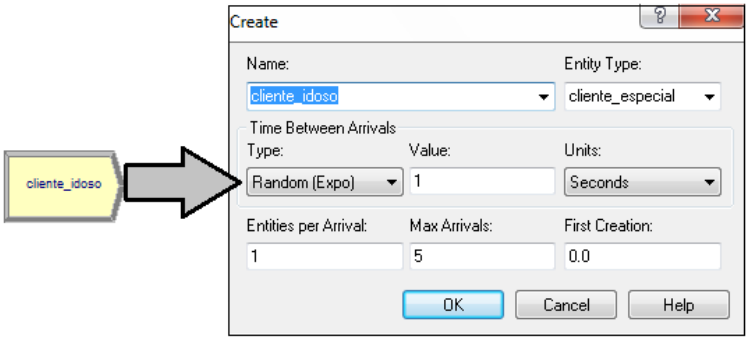

Figura 64: Create do cliente idoso.

A figura 64 apresenta outro Create, o nome é definido como cliente_idoso, nesta figura os campos são definidos igualmente à figura 63 (cliente), mas com uma diferença, os tipos das entidades deste Create são chamados de cliente_especial.

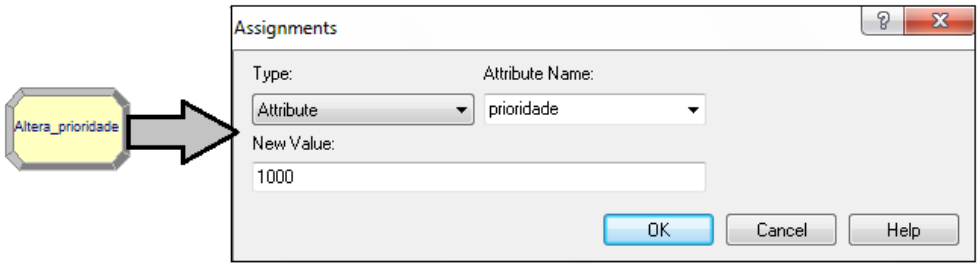

Figura 65: Atribuição de prioridade para o cliente idoso.

As entidades do tipo *cliente_especial* precisam ter sua prioridade maior que um *cliente_comum*, para isto foi necessário criar um atributo e defini-lo com o nome prioridade. Também foi criado o módulo **Assign** (responsável por alterar o atributo de prioridade) definindo como *altera_prioridade*, a configuração deste módulo é apresentado na figura 65, nesta é definido o nome do atributo que será alterado e o novo valor que o atributo irá receber, que para este caso foi declarado com o valor 1000.

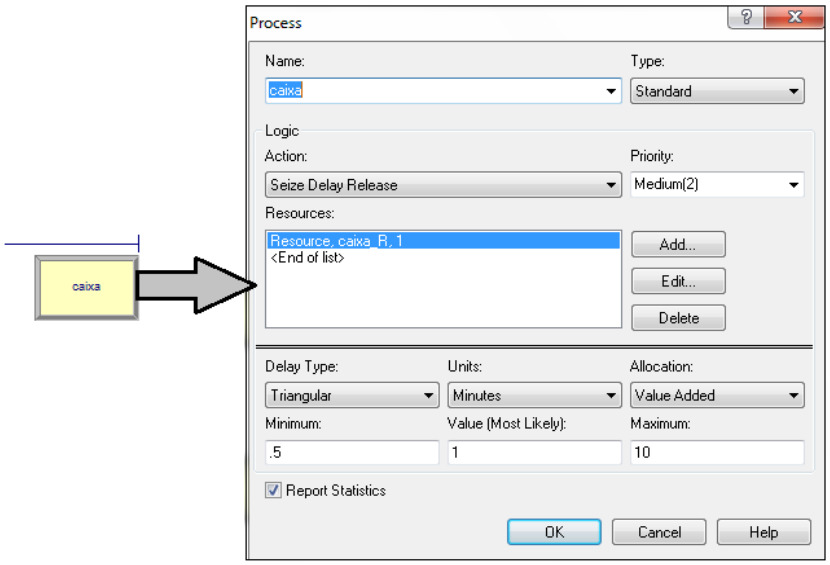

Figura 66: Módulo Process do caixa.

A figura 66 apresenta o módulo de **Process**, este processo é definido com o nome de caixa. Para o modelo em análise foi adicionado um novo recurso, definido como caixa_R e sua quantidade com o valor 1, que indica que somente uma unidade é necessária para atender uma entidade. O campo action é definido como Seize Delay Release, o atraso (delay) é definido em minutos variando de forma triangular: 0.5, 1 e 10, para valores: mínimo, mais provável e máximo, respectivamente.

Para modificar o regime da fila de atendimento do caixa é necessário clicar abrir a planilha *Queue*, esta é demonstrada na figura 67. Em *type* selecione *Highest Attribute Value* (A fila irá ordenar pelo valor maior do atributo), e em *atribute name* selecione prioridade (atributo criado anteriormente). Com esses procedimentos a ordenação da fila por prioridade será respeitada.

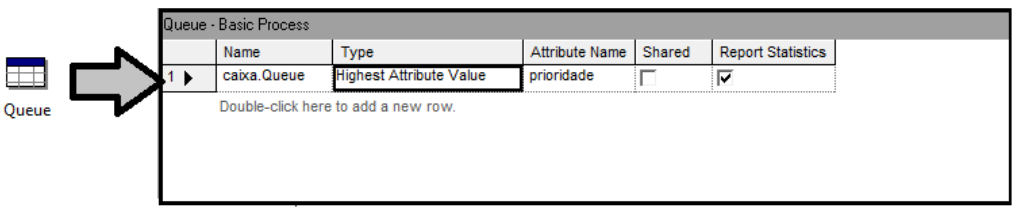

Figura 67: Alteração do regime da Fila .

Na figura 68 é apresentado o módulo **Decide** (decisão), neste ponto é definido que, quando uma entidade for passar pelo módulo ocorrerá uma verificação, caso seja um *cliente_especial* o resultado será verdadeiro e a entidade irá seguir para o módulo **Dispose** chamado *saída1*, caso contrário irá para o módulo *saída2.* O modelo é finalizado criando dois **Dispose**, um com o nome *saída1* e outro com o nome *saída2*.

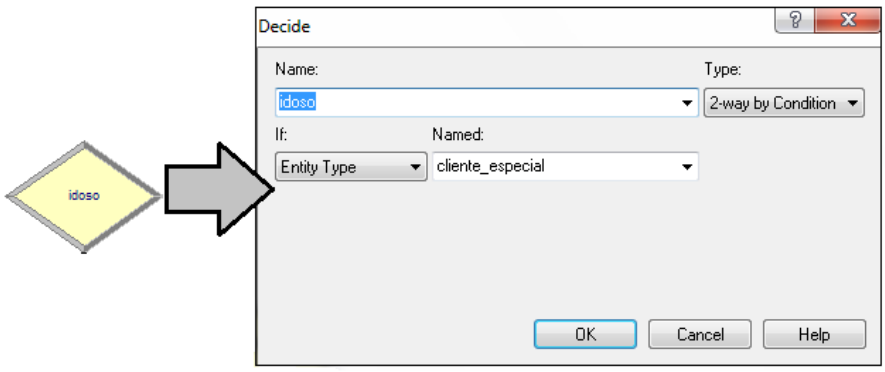

Figura 68: Módulo decide.

4.1.3 Análise da execução

O procedimento da análise de execução do modelo tem por função verificar se o modelo implementado satisfaz as exigências que foram impostas. O modelo proposto de uma fila de prioridade foi executado com um cenário de supermercado simplificado, foram definidos dez clientes que chegam ao caixa, sendo cinco, clientes idosos e cinco clientes comuns. Ao chegar ao caixa à primeira entidade foi de um cliente comum, como não havia ninguém na fila, este foi atendido. No tempo em que o cliente comum estava sendo atendido chegaram os demais clientes, e observou-se que alguns clientes comuns chegaram à frente dos clientes idosos, porém quando clientes idosos chegaram ao caixa à fila foi reorganizada.

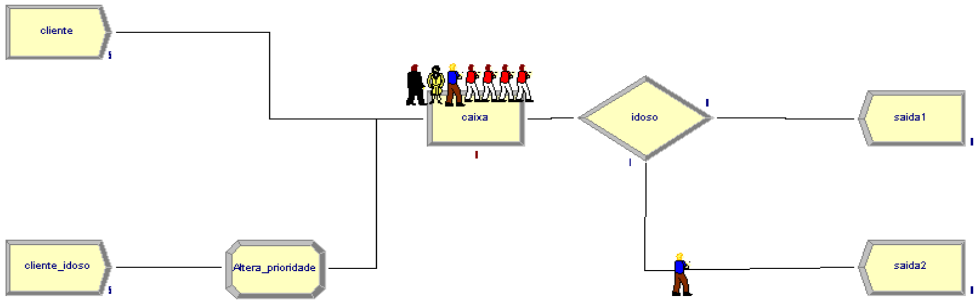

Figura 69: Execução do modelo de Fila.

A figura 69 apresenta a saída do primeiro cliente comum atendido no caixa, quando a fila estava vazia. Já a figura 70 ilustra a saída do segundo cliente do caixa, que neste caso é um idoso. O cliente comum foi o primeiro a ser atendido, pois quando ele chegou não havia ninguém na fila, e quando este estava sendo atendido no caixa chegaram os demais clientes e a fila do caixa foi reorganizada com os clientes idosos a frente dos clientes comuns.

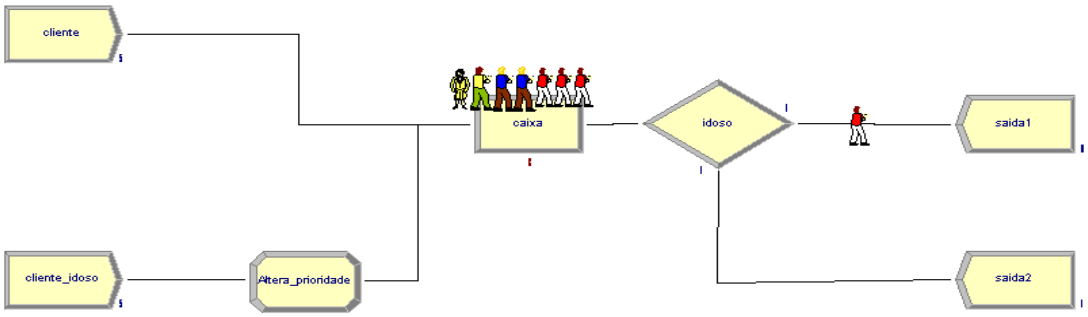

Figura 70: Execução do modelo de Fila.

A execução do modelo de fila representou de forma fiel o que foi proposto, ou seja, os processos de prioridade e reorganização do atendimento no caixa funcionaram de forma precisa.

4.1.4 Exemplo de Simulação do Protocolo Bit Alternado

A modelagem de simulação do Bit Alternado está representada na Figura 71, contendo os seguintes elementos de protocolo: o Transmissor, o Meio de Comunicação, o Receptor e uma Aplicação-Receptora.

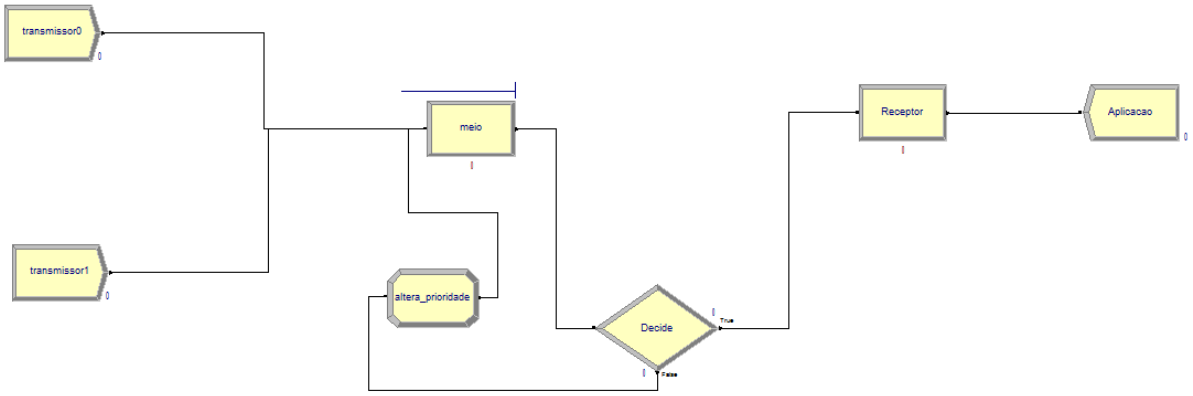

Figura 71: Modelo Bit Alternado no ARENA.

No sentido de simular o Protocolo do Bit Alternado, o **Transmissor** é representado na simulação por dois componentes **Create** do ARENA (*transmissor0* e *transmissor1*), nos quais, um dos componentes gera em um tempo constante uma mensagem “azul” (considerada como bit 0), enquanto o outro componente gera em um tempo constante uma mensagem “vermelha” (considerada como bit 1), simulando assim, a alternância do bit nas mensagens do protocolo. O funcionamento da simulação pode ser visualizado na figura 72, que representa a execução do modelo no simulador.

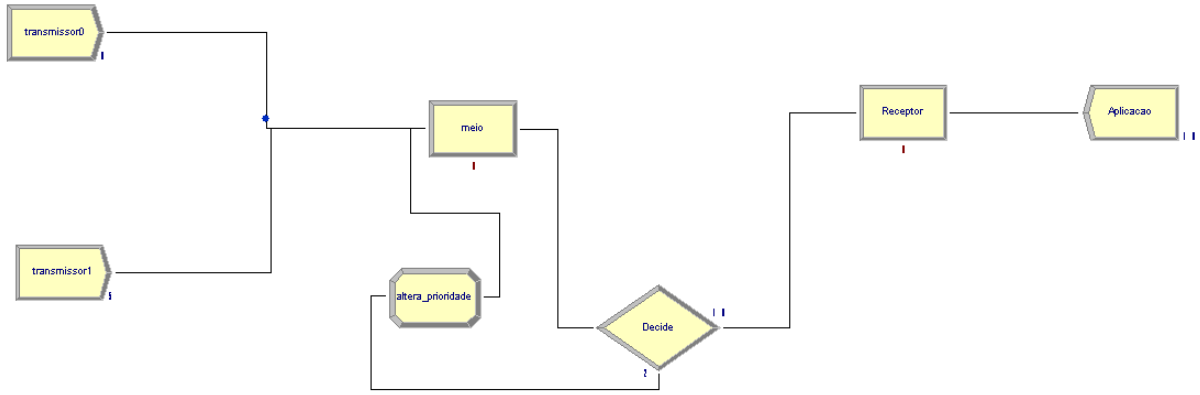

Figura 72: Execução do Modelo Bit Alternado no ARENA

O **Meio de Comunicação** é representado por um módulo **Process** do ARENA, no qual simula-se uma fila de mensagens. Cada mensagem na fila é repassada ao **Receptor**, que acumula mensagens em sua fila apropriada. Posteriormente, todas as mensagens da fila são repassadas à **Aplicação-Receptora**.

Eventualmente, um percentual é estipulado para representar um tempo finito, programado no componente **Decide** (ARENA), provoca uma mensagem poder ser perdida no **Meio de Comunicação**. Assim, existe um componente **Assign** do ARENA, que altera a prioridade do reenvio da mesma mensagem perdida, ao **Receptor**. Sem perda de generalidade, tal mensagem perdida é recolocada como primeiro elemento na fila do Meio de Comunicação, o qual será reenviado ao **Receptor**. Por questão da simplicidade do modelo aqui realizado, poderá surgir uma exceção, se nesse meio tempo de reenvio, a fila no **Meio de Comunicação** ficar vazia. Neste caso, essa mensagem perdida não será repassada na ordem correta (antes da próxima mensagem) ao **Receptor**.

4.2 Comparação entre CD++ e ARENA.

Com o estudo das duas ferramentas de modelagem e simulação CD++ e ARENA, é possível verificar suas peculiaridades. Por exemplo, ao comparar os modelos de Fila, verificou-se que, para fazer este tipo de modelagem de simulação, o ARENA foi mais prático, exigindo menos conhecimento técnico do desenvolvedor, pois esta já possui a estrutura de dados de uma fila. No CD++ a exigência do desenvolvedor é maior, além de conhecer as peculiaridades da ferramenta é preciso ter conhecimento de linguagem de programação, e seus paradigmas. Outro fator importante observado, é que a ferramenta CD++ apresentou-se menos restrita na questão de modelagem do que a ferramenta ARENA, ou seja, para modelagem de outros sistemas pode ser inviável devido à restrição dos módulos. A tabela 1 apresenta as características observadas utilizando as ferramentas CD++ e ARENA.

Características	CD++	ARENA
Uso de Linguagem de programação	Sim	Não
Facilidade de uso	Baixa	Alta
Flexibilidade na modelagem	Alta	Média
Reuso de modelos existentes	Sim	Não
Plataforma independente	Não	Sim
Possui mais de um modelo de simulação	Sim	Não

Tabela 1: Características das ferramentas CD++ e ARENA.

5. CONCLUSÃO

O presente trabalho abordou as principais funcionalidades das ferramentas CD++ e ARENA. Com isso é possível traçar um paralelo entre as duas ferramentas abordando a adequação do uso destas ferramentas para modelagem de sistemas.

A ferramenta CD++ é baseada em três modelos: Acoplado, Atômico e Celular, enquanto o ARENA oferece apenas uma opção de modelagem gráfica, com componentes pré-programados.

Para a simulação de sistemas através de modelagem atômica, foi observado que há necessidade do desenvolvedor e dominar a linguagem C++, enquanto ARENA, não exige o conhecimento de linguagem de programação. O conhecimento da linguagem gráfica de seus componentes é suficiente para um usuário iniciante. Isto pode ser considerado um fator positivo para desenvolvedores leigos, pois exigem um menor tempo no aprendizado da ferramenta. Tal fato ocorre na modelagem celular com a linguagem do CD++.

O ARENA apresenta uma interface bem amigável, enquanto o CD++ faz uso da IDE Eclipse muito conhecida por desenvolvedores. Entretanto, o que torna o uso do CD++ mais complexo é a necessidade de se conhecer uma serie de extensões de arquivos para compreender o uso da ferramenta (arquivo de log, arquivo de saída, arquivo de eventos, entre outros).

O estudo das duas ferramentas indicou que, para sistemas que utilizam estrutura de FILA e/ou simples processos (bancos, controle frotas, fábricas automatizadas) a ferramenta ARENA é mais viável, pois apresenta maior facilidade de modelar esses sistemas. Já o CD++ para modelagem de simples/médios sistemas ele pode ser inviável, pois o tempo de aprendizado e também, os recursos, às vezes simples, como uma estrutura de FILA, necessitam ser programados. Para simulação de protocolos de redes foi constatado que o ARENA não apresenta explicitamente facilidade de modelagem, precisando torná-lo específico para redes, enquanto o CD++ exige o conhecimento do modelo atômico, por ser programável e não ter módulos pré programados como no ARENA, que são mais flexíveis para o uso de modelagem de protocolos de redes.

REFERÊNCIAS BIBLIOGRÁFICAS

CASSANDRAS, C. G. Discrete Event Systems: **Modeling and Performance Analysis**, Irwin Publishing, 1993. (Harold Chestnut Prize, IFAC Best Control Engineering Textbook, 1999).

CD++. **Introduction to CD++.** 2008. Disponível em http://cell-devs.sce.carleton.ca. Acessado em: 11 de novembro de 2012.

EHRLICH, Pierre Jaques. **Pesquisa Operacional: Curso Introdutório. 4 ed**. São Paulo: Atlas, 1982.

FILHO, Clovis Perin. **Introdução à simulação de sistemas**. Campinas, SP: Editora da UNICAMP, 1995.

FILHO, Paulo Jose de Freitas. **Introdução à modelagem e simulação de sistemas**. Florianópolis, SC. Editora Visual Books, 2008.

FISHWICK, P. A. **Simulation Model Design and Execution**: Building Digital Worlds. Prentice Hall, 1995.

GARDNER, Martin. **Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life"**. 1970.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. São Paulo: Atlas, 1991.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D**. Introduction to automata theory, languages, and computation**, 3rd ed. Boston: Pearson/Addison–Wesley, 2007.

NANCE, R.E. **The time and state relationships in simulation modeling**. m. ACM, Abril 1981.

PARAGON. **Software de Simulação Arena**. 2008. Disponível em: http://www.paragon.com.br. Acesso em: 17 de dezembro de 2012.

RICHARDSON, Robert Jarryet al. **Pesquisa social: métodos e técnicas**. 3ª Ed. São Paulo: Atlas, 1999.

RODRÍGUEZ, Daniel A; WAINER, Gabriel A. **CD++ User'sGuide**. Buenos Aires: 1999.

SILVA, Ermes Medeiros da. et al. **Pesquisa Operacional: Programação Linear**. 2 ed. São Paulo: Atlas, 1996

WAINER, Gabriel A. **Discrete-event modeling and simulation: a practitioner's approach**.EUA: CRC Press, 2009.

WAINER, Gabriel; GIAMBIASI, Norbert. **Application of the Cell-DEVS Paradigm for Cell Spaces Modeling and Simulation**, SIMULATION 76:1, 2001, 22-39.

ZEIGLER, B. P. 1976. **Theory of modeling and simulation**, New York: Wiley-Interscience. 1976.

ZEIGLER, B. P.; PRAEHOFER, H.; KIM, T. G. **Theory of modeling and simulation**, 2nd. ed. New York: Academic Press. 2000.

APÊNDICE A

Modelagem e Simulação de Sistemas: Uma Abordagem das Ferramentas CD++ e ARENA

Antônio Francisco da Rosa Alves[1], Guilherme Cassol Espíndola[1]

[1]Departamento de Informática e Estatística - INE
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis, SC – Brasil

afralves@hotmail.com, cassol@outlook.com

Abstract. *This article describes the main aspects of modeling and simulation systems, providing a guide to how to use the CD + + tool and its different types of simulation models: Coupled Model, Model Cellular and Atomic. These aspects of modeling are compared, in general aspects of a simulation using another software simulator, called ARENA. Comparative conclusions of the simulation methods are shown emphasizing the simplicity of use, flexibility in modeling, reuse of existing models and platform independence, since CD + + IDE and requires an ARENA works alone.*

Resumo. *Este artigo descreve os principais aspectos de modelagem e simulação de sistemas, fornecendo um guia de como utilizar a ferramenta CD++ e seus diferentes tipos de modelos de simulação: Modelo Acoplado, Modelo Celular e Atômico. Esses aspectos de modelagem são comparados, em linhas gerais aos aspectos de uma simulação usando outro software simulador, chamado ARENA. Conclusões comparativas aos dos métodos de simulação são mostradas enfatizando a simplicidade de uso, a flexibilidade na modelagem, reuso de modelos existentes e a independência de plataformas, já que CD++ necessita de uma IDE e o ARENA funciona por si só.*

1. Introdução

Nos últimos anos, aplicações que executam tarefas com alto grau de complexidade vêm crescendo de forma exponencial. Avanços na tecnologia da computação tornam possível de automatizar tarefas a uma velocidade e qualidade desconhecida anteriormente. Essa tendência deve continuar em crescimento, exigindo ainda mais dos softwares qualidade, funcionalidade e confiabilidade quanto a seu desenvolvimento.

Modelos de simulação são desenvolvidos para detalhar o sistema a ser desenvolvido com as devidas qualidades e funcionalidades impostas, esses modelos são divididos em dois grupos, Simulação Discreta (são sistema que mudam de estado em instantes discretos, podendo acontecer com incrementos de tempo ou não), Simulação Contínua (são sistemas cujo estado muda continuamente no tempo, podendo ser contínuo no tempo ou discreto no

tempo, dependendo se os valores das variáveis dependentes estão sempre disponíveis em qualquer ponto do tempo simulado, ou apenas em pontos específicos). A simulação deve ser utilizada principalmente em dois momentos: quando problemas possuem alto grau de complexidade para o tratamento analítico, segundo quando a solução para um determinado problema possui alto custo. [Filho 1995].

O presente artigo possui como foco abordar os principais pontos de modelagem e simulação de sistemas, utilizando como ferramenta para essa avaliação o software CD++ em conjunto com a IDE de desenvolvimento Eclipse. Outro ponto a ser detalhado será o confronto entre os Softwares CD++ e ARENA (software de modelagem e simulação de sistemas), onde os métodos de simulação das ferramentas serão comparados, utilizando exemplos didáticos, no sentido de verificar a ferramenta mais adequada para determinado uso.

2. Conceitos de Modelagem e Simulação

A primeira técnica utilizada pelo ser humano para compreender melhor um ambiente é a experimentação, por exemplo, para fazer cerâmica, deve-se misturar água e barro, realizando ensaios diferentes até que a consistência desejada seja obtida. Experimentação era a única técnica forma que o ser humano tinha para aprender sobre o meio ambiente por milhares de anos, e ainda é um dos principais métodos utilizados na resolução de problemas [Wainer 2009].

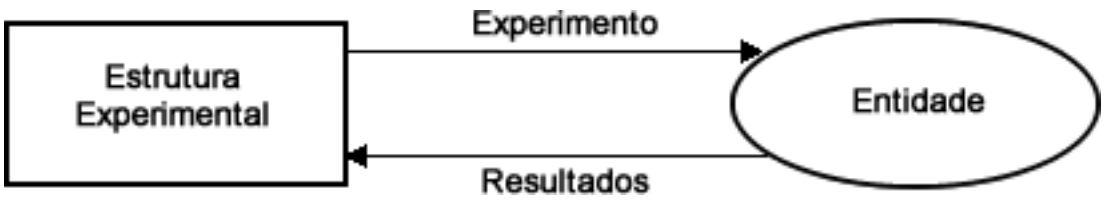

Figura 1. Resolução do problema através de experimentação [WAINER 2009].

A figura 1 apresenta um esquema básico para experimentação, nela há dois objetos em consideração: um é a entidade em estudo, e o outro a estrutura experimental (EF) que define as condições para tal experimentação. A estrutura experimental define não somente como se devem realizar experimentos na entidade, mas também como se obtém os resultados experimentais. Cada experiência seria um ensaio diferente para o experimento e os resultados dos experimentos devem determinar que ações devam ser tomadas para que o resultado esperado seja obtido [Wainer 2009].

Em alguns casos, a experimentação não é a solução mais recomendada devido à ética, os riscos (por exemplo, não se pode estudar a propagação de uma epidemia ou de evacuação de incêndio na sala de aula), ou o custo (não se podem estudar todas as configurações possíveis na sala de aula porque agendar um estudo com um grande número de indivíduos por

um longo tempo pode ser caro). Em outros casos, a experimentação não é possível de ser realizada [Wainer 2009].

A simulação pode ser vista como a manipulação de um modelo apresentando uma visão dinâmica, muito parecida com a realidade. A modelagem trabalha de forma paralela com a simulação, sendo que a modelagem é utilizada na criação de modelos, com a finalidade de representar de forma simplificada e explícita a realidade com algum propósito definido, onde os modelos são manipulados através da simulação de forma dinâmica, com o fluxo de entrada, processamento e saída de algum sistema ou objeto estudado. Os modelos de simulação podem ser divididos em dois grupos [Filho 1995]:

- **Simulação Discreta**: são sistemas que mudam de estado em instantes discretos (acontecimentos), essas mudanças podem acontecer com incrementos de tempos constantes ou incrementos de tempos variáveis. A variável tempo pode ser contínua ou discreta em determinado modelo, dependendo se as mudança discretas nas variáveis ocorrem em qualquer instante de tempo real ou em pontos predeterminados. A figura 2 apresenta um gráfico que relaciona uma variável dependente qualquer, através do tempo discreto [Filho 1995].

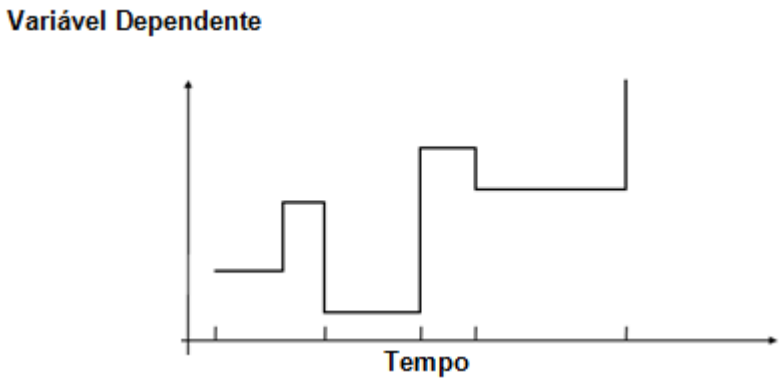

Figura 2. Gráfico Variável Dependente x Tempo.

- **Simulação Contínua**: são sistemas cujo estado muda continuamente no tempo, podem ser demonstrados utilizando como suporte as equações diferenciais. Um modelo contínuo pode ser tanto contínuo no tempo ou discreto no tempo, dependendo se os valores das variáveis dependentes estão sempre disponíveis em qualquer ponto do tempo simulado, ou apenas em pontos específicos. A figura 3 apresenta um gráfico que relaciona uma variável dependente qualquer, através do tempo contínuo [Filho 1995].

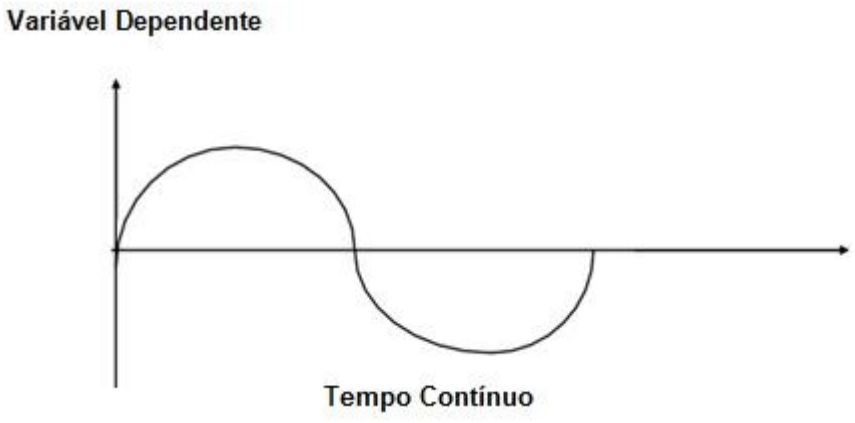

Figura 3. Variável dependente x Tempo Contínuo.

Principais aspectos que devem ser levados em consideração quanto ao uso de técnicas de simulação [Filho 1995]:

- Visualização: a disponibilidade de recursos computacionais para visualização dos resultados obtidos através da simulação (tabelas, diagramas, gráficos), bem como a situação do sistema durante a aplicação de um modelo;
- Custo: deve ser realizado um levantamento do objetivo e da disponibilidade de material suficiente para aplicação, bem como dos recursos físicos (hardware, software, equipamento em geral);
- Interferência: construção de modelos flexíveis para aplicação de mudanças, confrontado com o modelo de um sistema real. Esta característica possui como objetivo, gerar informação suficiente no apoio da definição dos resultados;
- Repetição: modelos de simulação devem ser construídos com objetivo de representar de forma mais fiel possível o sistema proposto, onde sua execução possa ser realizada n vezes a um custo muito baixo.

As vantagens do uso da simulação são muitas [Wainer 2009]:

- As decisões podem ser verificadas de forma artificial;
- O mesmo modelo podeser reutilizado várias vezes;
- Simulações são mais fáceis de criar e usar que várias técnicas analíticas, e precisam de menos simplificações;
- As regras usadas para definir o comportamento do modelo podem ser modificadas facilmente;
- O usuário pode interagir com o simulador, permitindo a análise das interações;
- A entidade original não é afetada pelo estudo, e pode continuar a ser utilizada.

2.1. Classificação de técnicas de Modelagem

A Definição da classificação dos métodos pode ajudar numa melhor compreensão dos dados. A seguinte classificação é fundamentada nos tipos de técnicas utilizadas na modelagem [Fishwick 1993]:

- **Modelagem conceitual**: Esta técnica é baseada na criação de um modelo conceitual informal que comunica com a natureza básica do processo. Ele fornece um vocabulário para o aplicativo (que pode ser ambíguo) e uma descrição geral da entidade a ser modelada [Wainer 2009];
- **Modelagem declarativa**: A técnica focaliza a evolução do modelo representado como estados (que descrevem o comportamento das entidades em estudo) e as transições entre eles [Wainer 2009];
- **Modelagem funcional**: O modelo é definido como uma "caixa preta" e a entrada é um sinal definido ao longo do tempo. A saída depende de uma função interna, e o modelo pode utilizar funções discretas ou contínuas [Wainer 2009];
- **Modelagem espacial**: Nesta técnica, noções de espaço são inclusos (isto é, a relação entre o tempo e o espaço é incluída no modelo) [Wainer 2009].

3. Introdução DEVS

A especificação de sistemas de eventos discretos (DEVS) é uma teoria de modelagem e simulação, que foi definida por B. Zeigler na década de 1970. O formalismo DEVS permite a descrição modular de modelos de eventos discretos que podem ser integrados através de uma abordagem hierárquica [Zeigler 1976; Zeigler, Kim, Praehofer, 2000].

De acordo com a teoria DEVS, o sistema de interesse é considerado como uma fonte de dados de comportamento para o estudo dentro de uma estrutura experimental (EF). A estrutura experimental é um conjunto restrito de elementos observados no sistema e as condições sob as quais são observadas. Estes dados são utilizados na criação de uma representação abstrata de um determinado sistema (um modelo). O modelo tenta reproduzir o comportamento do sistema de interesse sob as condições experimentais, através da utilização de um conjundo de instruções, regras ou equações matemáticas. A figura 4 apresenta essas entidades básicas na modelagem e simulação e suas relações [Zeigler 1976; Zeigler, Kim, Praehofer, 2000].

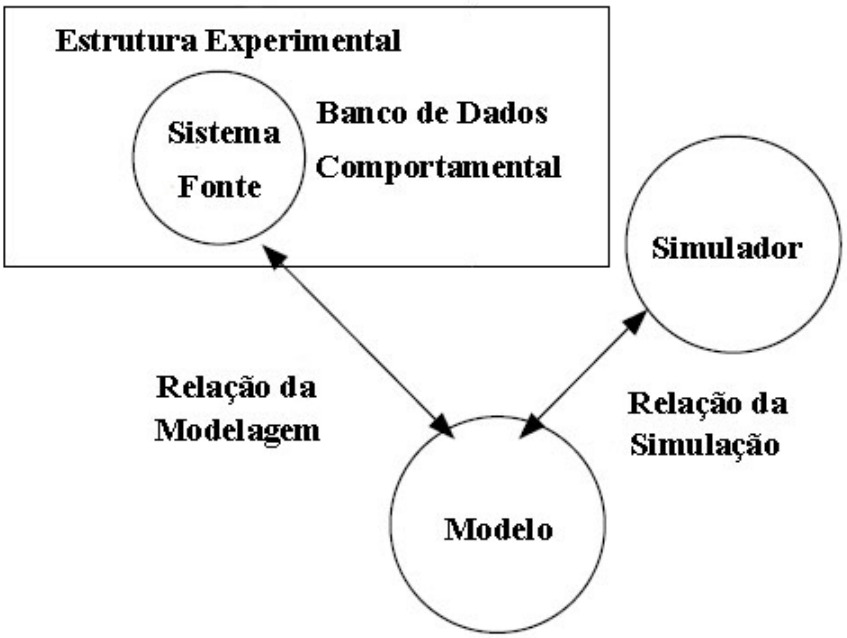

Figura 4. Entidades básicas em uma modelagem e simulação, com seus relacionamentos [ZEIGLER, 1976; ZEIGLER, KIM, PRAEHOFER, 2000].

As especificações formais proporcionam meios para manipulação matemática e permitem a independência da linguagem escolhida para implementar os modelos. O modelo é uma versão simplificada da realidade e sua estrutura. O modelo é construído considerando as condições de experimentação do sistema de interesse, incluindo as condições de trabalho do sistema real e de seu domínio de aplicação. Assim, o modelo é limitado à estrutura experimental em que foi desenvolvido (o que influência a sua construção, as tarefas de experimentação e de validação) [Wainer 2009].

Quando se executa as instruções de um modelo, ocorre certa redução na precisão dos resultados obtidos, devido aos recursos limitados numa linguagem de programação e um computador (incluindo a duração da simulação, disponibilidade de memória e precisão das variáveis envolvidas, etc.) [Wainer 2009].

DEVS foi criado para modelagem e simulação de eventos discretos de sistemas dinâmicos (DEDS), assim, ele define uma maneira de especificar sistemas em que os estados mudam, bem como na recepção de um evento de entrada ou devido à expiração em um determinado período de tempo [Wainer 2009].

4. Ferramenta de Modelagem e Simulação CD++

CD++ é uma ferramenta especializada na Modelagem de Eventos Discretos e simulação com base no formalismo DEVS. Seu funcionamento pode ser tanto em modo autônomo (CPU único) ou paralelo (através de uma interligação de máquinas dentro de uma rede). A ferramenta é utilizada como um plugin no software de desenvolvimento Eclipse, apresentando

um agrupamento de ferramentas que devem ser utilizadas no desenvolvimento de projetos, importação e exportação dos dados [Wainer 2009].

O aspecto essencial do CD++ é seu editor de acoplamento, que fornece para usuário a capacidade de manipular os eventos implementados, acoplando arquivos ao modelo criado. Funções básicas do editor são disponibilizadas, bem como comandos de teclado compatíveis. O editor suporta arquivos no formato XML para definição de um tipo especifico de modelo. O sistema foi projetado para fornecer um ambiente que atenda algum dos requisitos de desenvolvimento, como [Wainer 2009]:

- Criar e configurar um projeto;
- Exibir as perspectivas do projeto;
- Editar arquivos do modelo, incluindo palavras reservadas e destaque em palavras chaves;
- Apresentar informações e resultados.

A ferramenta é originalmente projetada para rodar em plataforma Windows, que limita sua utilização, pois devido à variedade de sistemas operacionais existentes no mercado, seu uso pode ser ineficiente em outro sistema operacional. O sistema operacional Linux, pode ser a melhor escolha para utilizar o software, além do Windows, dentre outros fatores positivos como sistema operacional distribuído, portabilidade o uso do compilador GCC como sua principal ferramenta de compilação no Linux é um dos principais fatores. O Compilador GCC é o compilador principal para a compilação dos modelos simulados. GCC possui como significado GNU Compiler Collection, sendo um conjunto de compiladores de linguagens de programação [Wainer 2009].

Integrado ao Eclipse, CD++ fornece uma interface unificada para todas as tarefas envolvidas na criação e atualização de modelos DEVS e modelos atômicos e acoplados, para análise dos resultados [Wainer 2009].

4.1. Definições dos Modelos

Os modelos acoplados são um conjunto de modelos atômicos combinados, este modelo especifica como as entradas e saídas dos componentes se conectam, de tal maneira a compor uma hierarquia [Wainer 2009].

Para os modelos acoplados existem quatro propriedades para configurar: componentes (utilizando a cláusula "components"), portas de saída (cláusula "out"), portas de entrada ("in") e as ligações entre os modelos (cláusula "link") [Rodríguez, Wainer 1999].

Um modelo atômico DEVS é um átomo funcional em um modelo, o qual não pode ser dividido em sub-componentes. Seu comportamento é descrito pela independência de implementação de funções matemáticas e símbolos [Wainer 2009].

Se a configuração para os modelos atômicos não é especificada, os valores padrões assumidos pelo desenvolvedor da classe serão utilizados [Rodríguez, Wainer 1999].

Modelos celulares são usados com um conjunto de parâmetros de certas características inerentes a eles. Esses parâmetros são [Rodríguez, Wainer 1999]:

- **Type: CELL:** Indica se o modelo celular é plano ou não;
- **Dim: (x0, x1, ..., xn):** Permite definir a dimensão de um modelo celular. Todos os valores **xi** devem ser inteiros. Se a cláusula Dim é usada, a invocação das cláusulas Width(largura) ou Height(altura) na mesmadefinição do modelo irá produzir um erro;
- **In:** Esta cláusula pode não ser definida, uma vez que a célula pode não ter portas de entrada conectadas com modelos externos;
- **Out:** Esta cláusula pode não ser definida, uma vez que a célula pode não ter portas de saída conectadas com modelos externos;
- **Link:** O mesmo que nos modelos acoplados, mas para fazer referência a uma célula que deve ser usado no nome do modelo acoplado em conjunto com (x1, x2,..., xn), sem deixar espaços;
- **Border :[WRAPPED | NOWRAPPED]:** Define o tipo de borda para o espaço celular. Por padrão, NOWRAPPED é utilizado;
- **Delay:** Ele especifica o tipo de atraso utilizado em todas as células do modelo;
- **DefaultDelayTime**: Define o atraso utilizado por padrão para os eventos externos (medido em milissegundos). O CD++ não impõe restrições sobre a criação do bairro, permitindo que seja possível utilizar mais do que um vizinho no modelo celular;
- **Initialvalue:** Representa o valor inicial para o espaço da célula. O símbolo representa o valor indefinido;
- **InitialCellsValue:** Especifica o nome do arquivo que contém os valores iniciais para as células de um modelo celular;
- **InitialCellsValue:** Pode ser usado com qualquer tipo de modelos celulares, mesmo com modelos bidimensionais;
- **InitialMapValue:** Especifica o nome do arquivo que contém um mapa de valores que vão ser usados como um estado inicial para um modelo celular;
- **LocalTransition**: Ele indica o nome do grupo que contém as regras que definem o local da função de computação para todas as células;
- **PortInTransition:** Permite definir um comportamento alternativo, quando uma mensagem chega do exterior para a porta de entrada especificado na célula (x1, x2, ..., xn) do modelo celular;
- **Zone**: Permite definir um comportamento alternativo para o grupo de células compreendidas dentro do faixa especificada. Cada faixa é definida como (x1, x2,..., xn) que descreve uma célula única, (x1, x2, ..., xn) .. (y1, y2, yn), descrevendo uma superfície de células, ou uma lista que pode combinar ambas (separação de cada elemento com um espaço em branco). Por exemplo: Zone: pothole{(10,10) .. (13, 13) (1,3)};

4.2. Arquivos de um projeto CD++

Na ferramenta CD++ ao criar um projeto de modelagem (CD++ BuilderProjectWizard), são necessários alguns arquivos para desenvolver uma modelagem e realizar simulações, estes arquivos podem ser ou não opcionais. Seguem abaixo as extensões e definições das funcionalidades de cada tipo de arquivo:

- **FileName.MA**: O arquivo de descrição do modelo;
- **FileName.EV (opcional):** arquivo de eventos externos;
- **Filename.log (opcional):** um arquivo de log gerado pelo simulador;
- **FileName.DRW (opcional):** um arquivo gerado pela ferramenta Drawlog(A ferramenta DrawLog permite representar graficamente a atividade do simulador para modelos celulares em cada instante de tempo, utilizando para isso os dados registrados no arquivo de log), onde os resultados podem ser vistos em uma matriz (disponível apenas em alguns modelos de celulares);
- **FileName.VAL** (opcional): Um arquivo com valores iniciais para o modelo;
- **FileName.BAT**: Um script usado para executar a simulação com os parâmetros corretos (apenas plataformas Windows, mas pode ser simplesmente modificado para ser usado em outras plataformas). Gera um arquivo de log;
- **FileNameDRW.BAT** (opcional): Um script usado para gerar o arquivo drw. ModelosDEVS escritos em C++;
- **NAME.CPP:** O arquivo com código-fonte em C++ que implementa os métodos do modelo atômico. Este arquivo é necessário para recompilar o simulador corretamente;
- **NAME.H:** O arquivo em C++ que é a interface do arquivo name.cpp. Este arquivo é necessário para recompilar o simulador propriamente.

4.3. Exemplo de projetos utilizando a ferramenta CD++

A Figura 5 apresenta a descrição do **modelo acoplado da Fila**, todo modelo acoplado (neste caso: Queue.ma) é composto por grupos de definições para os modelos de acoplamento (possibilitando também a configuração de modelos atômicos). A região destacada pelo retângulo apresenta o modelo acoplado e seus grupos de definições. O grupo *[top]* é obrigatório no modelo acoplado, *[Queue]* é outro grupo de definição, mas não obrigatório que representa o modelo atômico, para este projeto há necessidade de configurar quatro propriedades (cláusulas *components, link, in, out*).

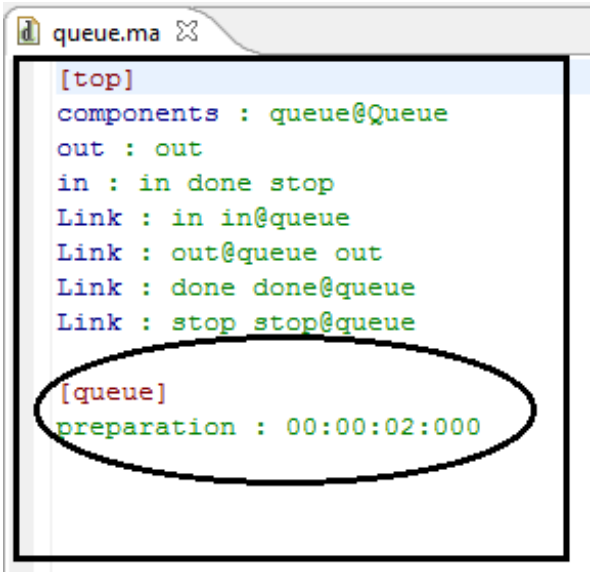

Figura 5. Definição dos componentes do modelo.

A cláusula **components** que descreve o(s) modelo(s) que compõe o modelo acoplado, neste caso é composta por um único modelo atômico chamado *queue@Queue* (*nomeModelo@ClasseDoModelo*). Quando um componente é declarado, é necessário criar o seu grupo de definição. A figura 5 foi marcada com uma elipse para identificar um grupo de definição, que é o modelo atômico *[Queue]*. Neste modelo é configurado o atributo *preparation*, responsável por indicar o tempo de preparação (*preparationTime*) para os elementos da fila, até que este seja enviado para fora, pela porta de saída.

A cláusula **Out** enumera as portas de saída, para esta modelagem se tem apenas uma porta de saída com o nome out. A cláusula **In** enumera as portas de entrada, para esta modelagem se tem 3 portas de entrada com o nome *in*, *done* e *stop*. A cláusula **Link** descreve o esquema interno e externo acoplado (Na figura 6 há apresentação do modelo em forma de diagrama).

O arquivo de modelagem **queue.ma** pode ser visualizado também em forma de diagrama, a figura 6, apresenta as declarações realizadas na figura 5 para o modelo acoplado Fila em formato de diagrama.

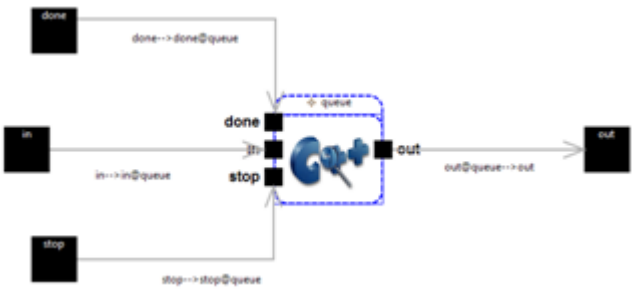

Figura 6. Componentes do modelo em diagrama.

5. Ferramenta de modelagem e simulação ARENA

O ARENA é um ambiente gráfico integrado de simulação, que contém todos os recursos para modelagem de processos, desenho e animação, análise estatística e análise de resultados [Paragon 2008]. O Software apresenta recursos de uma linguagem de simulação e um ambiente de trabalho e experimentação, que pode ser utilizado para testar o modelo e realizar a apresentação de seus resultados.

O software ARENA oferece componentes necessários para modelagem de simulação sem a necessidade de escrever linhas de código ou usar linguagem de programação, sendo que todo processo do desenvolvimento de modelos de simulação é baseado em fluxogramas, de maneira visual e interativa [Paragon 2008].

O foco principal da ferramenta é a criação templates, ou seja, uma coleção de objetos/ferramentas de modelagem, que possibilita ao usuário, detalhar o comportamento do processo em análise, através de respostas às perguntas pré-elaboradas, sem programação, de maneira visual e interativa [Paragon 2008].

Principais funcionalidades do Software [Paragon 2008]:

- Modelagem por Fluxogramas;
- Compatível com MS Office e Windows 7;
- Wizard/Assistente: Ajuda na criação de modelos;
- Bibliotecas: Extensa biblioteca de desenhos para interface animada.

5.1. Exemplo Bit Alternado no ARENA

O modelo simulado contem os seguintes elementos de protocolo: Transmissor, o Meio de Comunicação, o Receptor e uma Aplicação-Receptora.

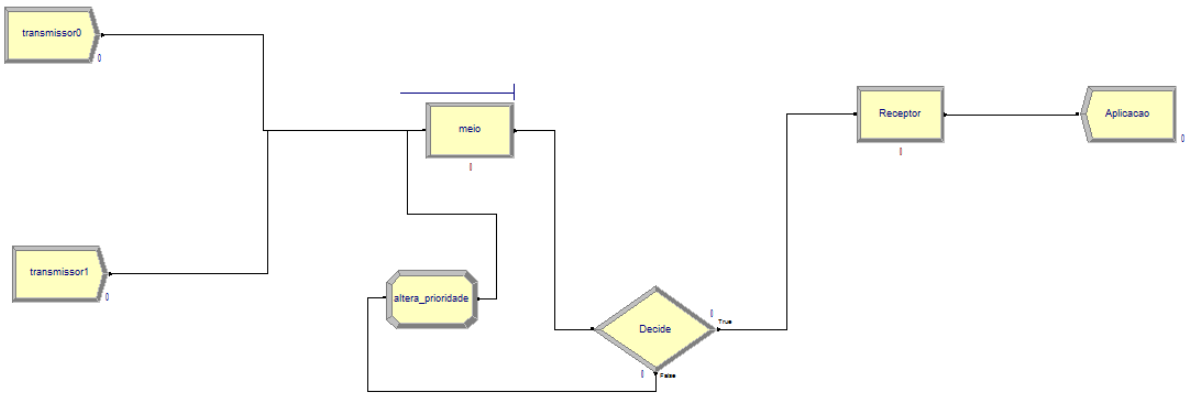

Figura 7. Modelo Bit Alternado.

No sentido de simular o Protocolo do Bit Alternanado, o **Transmissor** é representado na simulação por dois componentes **Create** do ARENA, nos quais, um dos componentes gera uma mensagem "azul" (considerada como bit 0), enquanto o outro componente gera uma

mensagem “vermelha” (considerada como bit 1), simulando assim, a alternância do bit nas mensagens do protocolo. O funcionamento da simulação pode ser visto na execução da figura 7 no simulador.

O **Meio de Comunicação** é representado por um componente **Process** do Arena, no qual simula-se uma fila de mensagens. Cada mensagem na fila é repassada ao **Receptor**, que acumula mensagens em sua fila apropriada. Posteriormente, todas as mensagens da fila são repassadas à **Aplicação-Receptora**.

Eventualmente, um percentual é estipulado para representar um tempo finito, programado no componente **Decide** (ARENA), provoca uma mensagem poder ser perdida no **Meio de Comunicação**. Assim, existe um componente **Assign** do Arena, que altera a prioridade do reenvio da mesma mensagem perdida, ao **Receptor**. Sem perda de generalidade, tal mensagem perdida é recolocada como primeiro elemento na fila do Meio de Comunicação, o qual será reenviado ao **Receptor**. Por questão da simplicidade do modelo aqui realizado, poderá surgir uma exceção, se nesse meio tempo de reenvio, a fila no **Meio de Comunicação** ficar vazia. Neste caso, essa mensagem perdida não será repassada na ordem correta (antes da próxima mensagem) ao **Receptor**.

6. Comparação entre CD++ e ARENA

Com o estudo das duas ferramentas de modelagem e simulação CD++ e Arena, é possível verificar suas peculiaridades.

Ao comparar as ferramentas e modelos utilizados verificou-se que, para alguns modelos a ferramenta arena foi mais prática, exigindo menos conhecimento técnico do desenvolvedor, pois o ARENA já possui módulos e estruturas de dados prontas, sendo assim não há necessidade do uso de linguagens de programação. No CD++ há exigência do desenvolvedor é maior, além de conhecer a peculiaridades da ferramenta é preciso ter conhecimento de linguagem de programação, e seus paradigmas. Outro fator importante observado, é que a ferramenta CD++ apresentou-se menos restrita na questão de modelagem do que a ferramenta Arena, ou seja, para modelagem de outros sistemas pode ser inviável devido à restrição dos módulos. A tabela 1 apresenta as características observadas utilizando as ferramentas CD++ e Arena.

Tabela 1. Características das ferramentas CD++ e ARENA.

Características	CD++	Arena
Precisa programar	Sim	Não
Facilidade de uso	Baixa	Alta
Flexibilidade na modelagem	Alta	Média
Reuso de modelos existentes	Sim	Não

Plataforma independente	Não	Sim
Possui mais de um modelo de simulação	Sim	Não

7. Conclusão

O presente artigo abordou as principais funcionalidades das ferramentas CD++ e ARENA. Com isso é possível traçar um paralelo entre as duas ferramentas abordando a adequação do uso destas ferramentas para modelagem de sistemas.

A ferramenta CD++ é baseada em três modelos: Acoplado, Atômico e Celular, enquanto o ARENA oferece apenas uma opção de modelagem gráfica, com componentes pré-programados.

Para a simulação de sistemas através de modelagem atômica, foi observado que há necessidade do desenvolvedor e dominar a linguagem C++, enquanto ARENA, não exige o conhecimento de linguagem de programação. O conhecimento da linguagem gráfica de seus componentes é suficiente para um usuário iniciante. Isto pode ser considerado um fator positivo para desenvolvedores leigos, pois exigem um menor tempo no aprendizado da ferramenta.

O ARENA apresenta uma interface bem amigável, enquanto o CD++ faz uso da IDE Eclipse muito conhecida por desenvolvedores. Entretanto, o que torna o uso do CD++ mais complexo é a necessidade de se conhecer uma serie de extensões de arquivos para compreender o uso da ferramenta (arquivo de log, arquivo de saída, arquivo de eventos, entre outros).

O estudo das duas ferramentas indicou que, para sistemas que utilizam estrutura de FILA e/ou simples processos (bancos, controle frotas, fábricas automatizadas) a ferramenta ARENA é mais viável, pois apresenta maior facilidade de modelar esses sistemas. Já o CD++ para modelagem de simples/médios sistemas ele pode ser inviável, pois o tempo de aprendizado e também, os recursos, às vezes simples, como uma estrutura de FILA, necessitam ser programados. Para simulação de protocolos de redes foi constatado que o ARENA não apresenta explicitamente facilidade de modelagem, precisando torná-lo específico para redes, enquanto o CD++ exige o conhecimento do modelo atômico, por ser programável e não ter módulos pré programados como no ARENA, que são mais flexíveis para o uso de modelagem de protocolos de redes.

8. Referências

FILHO, Clovis Perin. Introdução à simulação de sistemas. 1995. 159f. Campinas, SP: Editora da UNICAMP.

FILHO, Paulo Jose de Freitas. Introdução à modelagem e simulação de sistemas. Florianópolis, SC. Editora Visual Books, 2008.

FISHWICK, P. A. 1995. Simulation model design and execution: Building digital worlds. EnglewoodCliffs, NJ: Prentice Hall.

PARAGON. Software de Simulação Arena. 2008. Disponível em: <www.paragon.com.br>. Acesso em: 17dez. 2012.

RODRÍGUEZ, Daniel A; WAINER, Gabriel A., 1999.CD++ User'sGuide.

WAINER, G.; GIAMBIASI, Norbert, Application of the Cell-DEVS Paradigm for Cell Spaces Modeling and Simulation, SIMULATION 76:1, 2001, 22-39.

ZEIGLER, B. P. 1976. *Theory of modeling and simulation*, New York: Wiley-Interscience.

ZEIGLER, B. P.; PRAEHOFER H.; KIM T. G. 2000. *Theory of modeling and simulation*, 2nd. ed. New York: Academic Press.