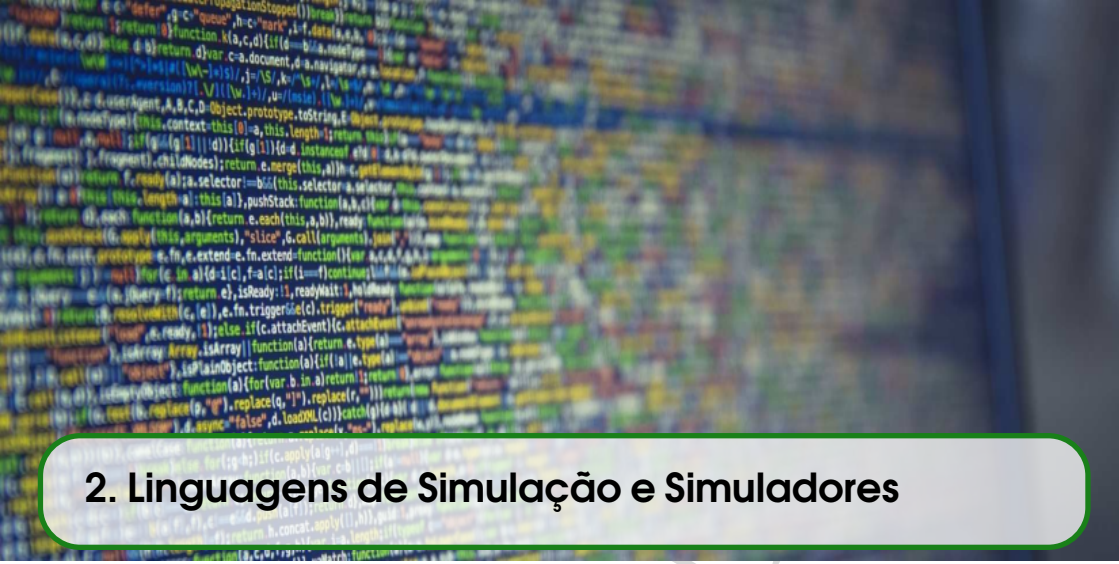

2. Linguagens de Simulação e Simuladores

Objetivos de aprendizagem:

- Conhecimentos:
  - Compreender conceitos, classificações e estrutura de linguagens de simulação.
  - Diferenciar linguagens de simulação e simuladores.
  - Compreender as vantagens, aplicações e limitações de diferentes linguagens de simulação e de simuladores.
  - Classificar e comparar adequadamente diferentes linguagens de simulação e simuladores.
  - Compreender a estrutura interna de um simulador.
- Habilidades:
  - Fazer download e instalar diferentes simuladores.
  - Utilizar de forma básica diferentes simuladores.

2.1 Introdução

Um modelo de simulação por computador é uma entidade abstrata que descreve o comportamento de um sistema de interesse. Ele pode ser projetado de acordo com diferentes metodologias de modelagem e implementado em um computador por meio de uma linguagem de programação. Muitas linguagens de programação específicas foram desenvolvidas para facilitar essa implementação. Elas são formalmente designados como linguagens de modelagem. A execução de um modelo de computador é chamada de simulação ou fase experimental. Alguns ambientes modernos de simulação em software fornecem extensões para linguagens de modelagem que simplificam tarefas comuns como identificação de parâmetros, análise de sensibilidade, validação e até a conexão com fontes de dados externas e outros programas de software. Assim, linguagens específicas usadas para descrever, implementar e até projetar os experimentos subsequentes são chamadas de linguagens de modelagem e simulação ou, simplesmente, linguagens de simulação. Usamos "linguagem de modelagem"e "linguagem de simulação"como termos sinônimos em todo o texto, apesar de eles não serem formalmente iguais. Utilizaremos a seguinte classificação nesta obra:

- **Linguagem de programação**: Liguagem genérica para desenvolvimento de software, como C++, Java, Python e muitas outras. Pode-se criar modelos e realizar simulação utilizando diretamente linguagens de programação, mas todo o mecanismo de suporte à modelagem e simulação (e análise dos resultados) precisa ser desenvolvido, o que pode envolver muito tempo e esforço. Sua vantagem costuma ser a familiaridade que que analistas devem possuir com alguma linguagem de programação existente.
- **Linguagem de simulação**: Linguagem específica para criação de modelos e/ou para simulação. Precisa ser interpretada por algum simulador para utilizar os recursos de modelagem e simulação existentes (previamente desenvolvidos usando uma linguagem de programação). Requer bem menos esforço do analista para especificar um modelo, simulá-lo e analisar os resultados da simulação.
- **Biblioteca de simulação**: Um conjunto de componentes (funções, classes, módulos) implementados numa linguagem de programação e que fornecem os serviços de suporte à modelagem, simulação e análise dos resultados. Esses componentes são uma extensão de uma linguagem de programação, e não uma nova linguagem de simulação. Permite que qualquer programa possa incluir a funcionalidade de simulação, sem necessariamente ser um simulador.

Nenhuma linguagem de simulação é capaz de simular todo e qualquer sistema de diferentes domínios, pois eles são muito diferentes entre si. Assim, existem muitas dezenas linguagens e cada uma foca em determinadas técnicas de modelagem e simu-

lação e em determinados domínios de aplicação. Nosso objetivo é apresentar muito rapidamente algumas linguagens de características consideravelmente diferentes e então focar em duas ou três linguagens com características úteis no contexto desta obra. Deve-se observar, entretanto, que muitas vezes a linguagem de simulação está integrada a um simulador, ou mesmo o simulador ou ambiente de simulação permite a descrição dos modelos apenas de forma gráfica, e não por uma "linguagem de simulação". Por isso são apresentadas seções distintas para linguagens e para simuladores.

Alguns fatores que devem ser considerados para a escolha de uma linguagem de simulação ou simulador para uma empresa ou estudo específico, incluem:

- Compatibilidade com os sistemas computacionais existentes. A linguagem ou simulador executa nesses computadores e sistemas operacionais?
- Existência de suporte e documentação. A linguagem ou simulador possui suporte ao usuário, tem manuais, documentação, exemplos, etc?
- Adequação aos sistemas e problemas que serão modelados. A linguagem ou simulador simulam os problemas de interesse? São adequados a eles e projetados para esses domínios de aplicação?
- Custos para obter, instalar, manter e atualizar os softwares.
- Dificuldade de aprendizado. A equipe já conhece a linguagem ou simulador e tem experiência prática com ele? Quão difícil será aprender a usá-lo no grau necessário para crias os modelos de interesse?
- Eficiência computacional. A linguagem ou simulador é computacionalmente eficiente e tira proveito das características arquiteturais dos computadores? Executam em paralelo ou em ambiente distribuído? Quanto tempo levará a simulação dos modelos de interesse?
- Ferramentas existentes. A linguagem ou simulador possui ferramentas acessórias para coleta e tratamento de dados, geração de cenários e experimentos, análise estatística dos resultados, depuração, diagnóstico de erros, etc?
- Custo/Benefício. Todos os custos e dificuldades envolvidos com a linguagem ou simulador valem a pena, se comparados com os benefícios que serão alcançados?

2.2 Linguagens de Simulação

Uma linguagem de simulação de computador é usada para descrever o comportamento de um modelo e a operação de uma simulação em um computador. Existem dois tipos principais de simulação: evento contínuo e evento discreto, embora as linguagens mais modernas possam lidar com combinações mais complexas. Linguagens de

simulação geralmente provêm pelo menos as seguintes funcionalidades:

- especificação do modelo,
- geração de números aleatórios e variáveis aleatórias,
- mecanismos de avanço do tempo,
- coleta de dados para análise,
- análise de dados coletados,
- verificação e diagnóstico de erros.

Existem várias dezenas de linguagens de simulação ou mesmo linguagens de programação usadas para modelagem e simulação, e apresentamos a seguir uma pequena lista com a descrição básica de algumas delas. As linguagens de maior interesse serão melhor detalhadas em subseções específicas.

DYNAMO (DYNAmic MOdels) é uma linguagem de simulação historicamente importante. Era originalmente projetada para dinâmica industrial, mas logo foi estendida a outras aplicações, incluindo estudos de população e planejamento urbano. O DYNAMO foi desenvolvido no final dos anos 1950, mas desde então caiu em desuso. Além do seu impacto público nas questões ambientais levantadas pela controvérsia sobre os limites ao crescimento, o DYNAMO foi influente na história da simulação de eventos discretos, apesar de ser essencialmente um pacote de simulação contínua especificado por meio de equações das diferenças. Alguns dizem que DYNAMO abriu oportunidades para modelagem por computador, mesmo para usuários com sofisticação matemática relativamente baixa.

Modelica é uma linguagem de modelagem declarativa e orientada a objetos, com vários domínios, para modelagem orientada a componentes de sistemas complexos, como sistemas contendo subcomponentes mecânicos, elétricos, eletrônicos, hidráulicos, térmicos, de controle, energia elétrica ou orientados a processos. A linguagem Modelica gratuita é desenvolvida pela *Modelica Association*, sem fins lucrativos. A *Modelica Association* também desenvolve a Biblioteca Padrão da Modelica gratuita, com milhares de componentes genéricos e funções em vários domínios.

NetLogo é uma linguagem de programação e um ambiente de desenvolvimento integrado (IDE) para modelagem e é baseado em agentes. O NetLogo foi projetado para diferentes públicos, em particular: ensinar crianças na comunidade educacional e especialistas em domínio sem experiência em programação para modelar fenômenos relacionados. O ambiente NetLogo permite a exploração de fenômenos emergentes. Ele vem com uma extensa biblioteca de modelos, incluindo modelos em vários domínios, como economia, biologia, física, química, psicologia, dinâmica de sistemas. O NetLogo permite interação modificando interruptores, controles deslizantes, seletores, entradas e outros elementos da interface. O NetLogo é de código aberto, está disponível gratuitamente no site do NetLogo e está em uso em uma ampla variedade

de contextos educacionais, da escola primária à pós-graduação.

SIM.JS é uma biblioteca de simulação de eventos discretos baseada em eventos, baseada no JavaScript padrão. A biblioteca foi escrita para permitir a simulação em navegadores padrão utilizando a tecnologia da web. SIM.JS suporta entidades, recursos (instalações, buffers e lojas), comunicação (via temporizadores, eventos e mensagens) e estatísticas (com estatísticas de séries de dados, séries temporais e população). A distribuição SIM.JS contém tutoriais, documentação detalhada e um grande número de exemplos, e foi lançado como software de código aberto sob a licença LGPL, sendo a primeira versão em janeiro de 2011.

SimPy é linguagem uma estrutura de simulação de eventos discretos baseada em processo, baseada no Python padrão. Seu distribuidor de eventos é baseado nos geradores do Python e também pode ser usado para redes assíncronas ou para implementar sistemas multi-agente (com comunicação simulada e real). Os processos no SimPy são funções simples de gerador de Python e são usados para modelar componentes ativos como clientes, veículos ou agentes. O SimPy também fornece vários tipos de recursos compartilhados para modelar pontos de congestionamento de capacidade limitada (como servidores, balcões de verificação e túneis), e também fornece recursos de monitoramento para ajudar na coleta de estatísticas sobre recursos e processos. A distribuição SimPy contém tutoriais, documentação detalhada e um grande número de exemplos. O SimPy foi lançado como software de código aberto sob a licença MIT, sendo a primeira versão foi lançada em dezembro de 2002.

GPSS (General Purpose Simulation System) é uma linguagem de programação de uso geral de simulação de tempo discreta, em que um relógio de simulação avança em etapas discretas. Foi desenvolvida pela IBM no começo da década de 1960. Um sistema é modelado à medida que as entidades entram no sistema e são passadas de um serviço (representado por blocos) para outro. É usado principalmente como uma linguagem de simulação orientada ao fluxo de processo e é particularmente adequada para sistemas como uma fábrica. a listagem 2.1 mostra um modelo de simulação descrito usando a linguagem GPSS. Nesse modelo, "comandos"como GENERATE, QUEUE, SEIZE, DEPART, etc representam os serviços ou blocos que compõem o modelo. A ordem desses blocos representa o fluxo pelo qual as entidades são submetidas (geralmente começam com GENERATE, que geram entidades e encerram com TERMINATE, que exclui entidades).

```
1  SIMULATE              ; Define model
2  * Model segment 1
3  GENERATE 18,6         ; Customer arrive every 18+-6 mn
4  QUEUE    Chairs       ; Enter the line
5  SEIZE    Joe          ; Capture the barber
6  DEPART   Chairs       ; Leave the line
```

```
7   ADVANCE 16,4          ; Get a hair cut in 16+-4 mn
8   RELEASE Joe           ; Free the barber
9   TERMINATE             ; Leave the shop
10  *  Model segment 2
11  GENERATE 480          ; Timer arrives at time = 480 mn
12  TERMINATE 1           ; Shut off the run
13  *  Control cards
14  START    1            ; Start one run
15  END                   ; End model
```

Listing 2.1: Modelo de barbearia com GPSS

Atualmente a ferramenta JPSS (*Java General Purpose Simulation System*) é usada para ensinar essa linguagem.

As subseções abaixo apresentam com um pouco mais de detalhes algumas linguagens de simulação de maior interesse nesta obra.

2.2.1 Modelica

Modelica é uma linguagem orientada a objetos disponível gratuitamente para modelagem de sistemas físicos grandes, complexos e heterogêneos. Do ponto de vista do usuário, os modelos são descritos por esquemas, também chamados de diagramas de objetos. Exemplos são mostrados na próxima figura:

Um esquema consiste em componentes conectados, como um resistor ou um cilindro hidráulico. Um componente possui "conectores"(geralmente também chamados de "portas") que descrevem as possibilidades de interação, como um pino elétrico, um flange mecânico ou um sinal de entrada. Ao desenhar linhas de conexão entre conectores, um sistema físico ou bloco de diagrama é construído. Internamente, um componente é definido por outro esquema ou no nível "inferior", por um descrição baseada em equações do modelo na sintaxe do Modelica.

A linguagem Modelica é uma descrição textual para definir todas as partes de um modelo e estruturar componentes do modelo em bibliotecas, chamados pacotes. Um ambiente de simulação Modelica apropriado é necessário para editar e navegar graficamente em um modelo Modelica (interpretando as informações que definem um modelo Modelica) e para realizar simulações de modelo e outras análises. Informações sobre esses ambientes estão disponíveis em www.modelica.org/tools. Basicamente, todos os elementos da linguagem Modelica são mapeados para equações diferenciais, algébricos e discretas. Não há elementos de linguagem para descrever equações diferenciais parciais diretamente, embora alguns tipos de equações diferenciais parciais discretizadas podem ser razoavelmente definidas, por exemplo, com base no método do volume finito, e existem bibliotecas Modelica para importar resultados de

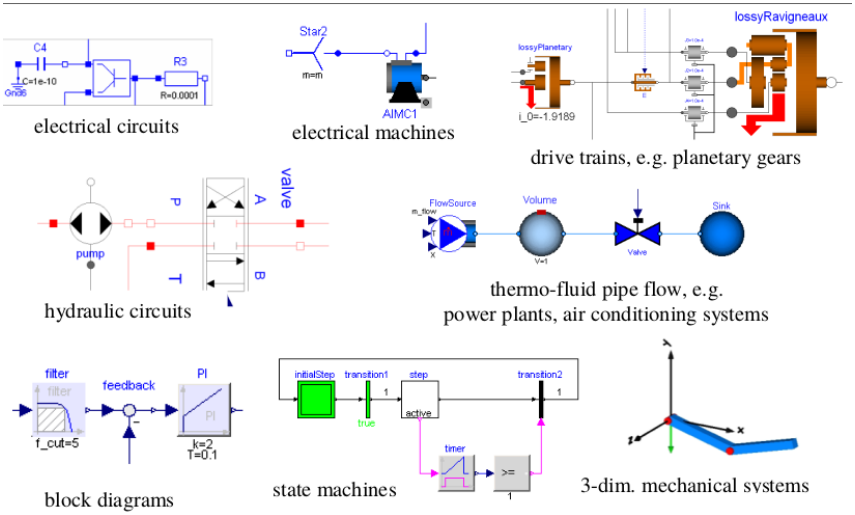

Figura 2.1: Exemplos de diagramas de objetos em Modelica

programas de elementos finitos.

Para dar uma introdução ao Modelica, consideraremos a modelagem de um circuito elétrico simples como mostrado na figura 2.2.

O sistema pode ser dividido em um conjunto de componentes elétricos conectados. Há uma fonte de tensão, dois resistores, um indutor, um capacitor e um ponto de aterramento. Modelos desses componentes estão disponíveis nas bibliotecas de modelos e, usando um editor de modelos gráficos, pode definir um modelo desenhando um diagrama de objetos muito semelhante ao diagrama de circuitos mostrado acima, posicionando ícones que representam os modelos dos componentes e desenhando conexões. Uma descrição Modelica do circuito completo poderia ser o seguinte (2.2):

```
1  model circuit
2     Resistor R1(R=10);
3     Capacitor C(C=0.01);
4     Resistor R2(R=100);
5     Inductor L(L=0.1);
6     VsourceAC AC;
7     Ground G;
8  equation
9     connect (AC.p, R1.p);
10    connect (R1.n, C.p);
11    connect (C.n, AC.n);
```

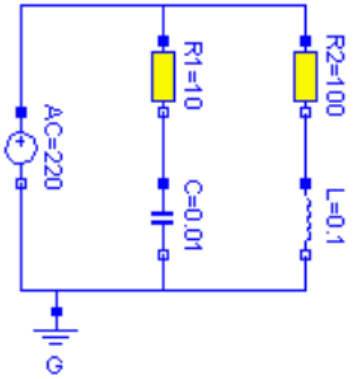

Figura 2.2: Circuito eletrônico a ser modelado

```
12    connect (R1.p, R2.p);
13    connect (R2.n, L.p);
14    connect (L.n, C.n);
15    connect (AC.n, G.p);
16 end circuit;
```

Listing 2.2: Modelo do circuito eletrônico com Modelica

Para maior clareza, a definição do layout gráfico do diagrama de composição (aqui: diagrama de circuitos) não é mostrado, embora geralmente esteja contido em um modelo Modelica anotações (que não são processadas por um tradutor do Modelica e usadas apenas por ferramentas). Um modelo composto desse tipo especifica a topologia do sistema a ser modelado. Especifica o componentes e as conexões entre os componentes. A declaração
`Resistor R1(R=10);` declara que um componente `R1` é da classe `Resistor` e define o valor padrão da resistência, `R`, ao valor 10. As conexões especificam as interações entre os componentes. O elemento `connect` é um operador especial que gera equações levando em consideração que tipo de quantidades são envolvidos.

Um conector deve conter todas as quantidades necessárias para descrever a interação. Para componentes elétricos precisamos das quantidades de tensão e corrente para definir a interação via um fio. Os tipos para representá-los são declarados como
`type Voltage = Real (unit = "V");`
`type Current = Real (unit = "A");`
onde `Real` é o nome de um tipo de variável predefinido. Uma variável real possui um conjunto de atributos como como unidade de medida, valor inicial, valor mínimo e

máximo. Aqui, as unidades de medida são definido como as unidades SI (Sistema Internacional).

No Modelica, o elemento estruturador básico é uma classe. Existem sete classes restritas com nomes específicos, como `model`, `type` (uma classe que é uma extensão de classes internas, como `Real`, ou de outros tipos definidos), `connector` (uma classe que não possui equações e pode ser usado em conexões).

O conceito de classes restritas é vantajoso porque o modelador não precisa aprender vários conceitos diferentes, mas apenas um: o conceito de classe. Todas as propriedades de uma classe, como sintaxe e semântica de definição, instanciação, herança e genérico são idênticas a todos os tipos de classes restritas. Além disso, a construção dos tradutores Modelica é simplificada consideravelmente porque somente a sintaxe e a semântica de uma classe precisam ser implementadas junto com algumas verificações adicionais em classes restritas. Os tipos básicos, como `Real` ou `Integer`, são construídos nas classes de tipos, ou seja, eles têm todas as propriedades de uma classe e os atributos desses tipos básicos são apenas parâmetros da classe.

Se o modelo estrutural apresentado em 2.2 fosse representado como um diagrama de blocos, a estrutura do modelo seria bem diferente, como apresentado na figura **??**. O diagrama de blocos é equivalente a um conjunto de atribuição envolvendo derivadas (equações diferenciais). De fato, a lei de Ohm é usada de duas maneiras diferentes neste circuito, uma vez resolvendo para i (corrente) e outra vez resolvendo para u (tensão). Este exemplo mostra claramente os benefícios da modelagem não causal (como em Modelica) em comparação à modelagem causal (como em Simulink).

2.2.2 SIMAN

O SIMAN (SIMulation ANalysis) é uma linguagem de simulação de uso geral para modelagem combinada sistemas discretos e contínuos e se baseou no GPSS. O framework de modelagem do SIMAN permite modelos de componentes baseados em três orientações de modelagem distintas a serem combinadas em um único modelo de sistema. Para sistemas discretos pode ser usada orientação a processo ou orientação a eventos para descrever o modelo. Sistemas contínuos são modelados com álgebra ou equações diferenciais. Uma combinação dessas orientações podem ser usadas para criar modelos híbridos (discretos e contínuos).

A estrutura de modelagem do SIMAN é baseada nos conceitos teóricos dos sistemas desenvolvidos por Zeigler (1976). Dentro dessa estrutura, destaca-se uma distinção fundamental entre o modelo do sistema e o plano experimental. O modelo do sistema define as características estáticas e dinâmicas do sistema. O plano experimental define as condições do experimento sob as quais o modelo é executado

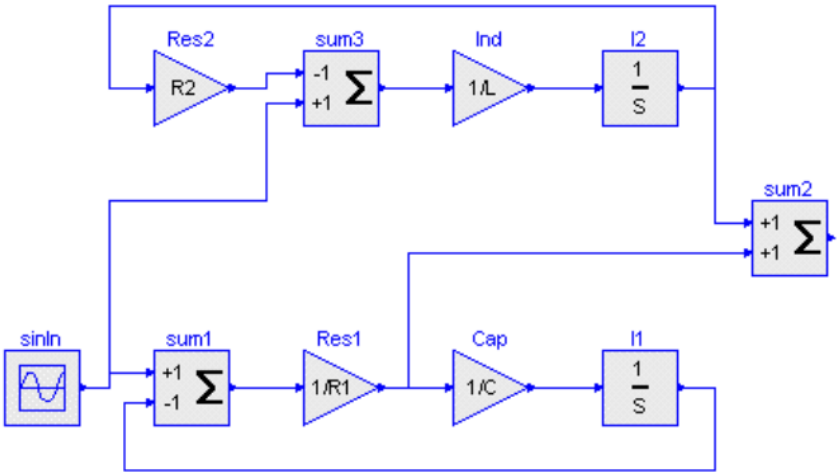

Figura 2.3: Modelo causal equivalente ao circuito eletrônico apresentado anteriormente

para gerar dados de saída específicos. Ao separar a estrutura do modelo e a estrutura experimental em dois elementos distintos, diferentes experimentos de simulação podem ser realizados alterando apenas a estrutura experimental, e a estrutura do modelo permanece a mesma.

Embora o SIMAN permita que os modelos possam ser desenvolvido usando orientação a processo, eventos ou contínuos, focamos na orientação do processo em que os modelos são construído como diagramas de blocos. Essa orientação é a mais adequado para modelar a maioria dos sistemas de manufatura. Esses diagramas são fluxogramas que descrevem o fluxo de entidades através do sistema. As formas do símbolo do diagrama de blocos indicam sua função. O sequenciamento de blocos é representado por setas que controlam o fluxo de entidades de bloco para bloco ao longo do diagrama (processo).

Há dez tipos de blocos básicos no SIMAN, que são OPERATION, HOLD, TRANSFER, QUEUE, STATION, BRANCH,PICKQ, QPICK, SELECT, e MATCH. Todos os tipos básicos de blocos possuem operandos (parâmetros) que controlam a função do bloco. Por exemplo, o bloco CREATE possui operandos que informam o tempo entre as chegadas do lote, o número de entidades por chegada de lotes, o número de entidades por lote e o número máximo de lotes a serem criados. Opcionalmente, pode ser atribuído aos blocos uma etiqueta (label), um ou mais modificadores de bloco e uma linha de comentário.

Um modelo pode ser definido em de duas formas equivalentes referidas como diagrama ou declaração. O *diagrama do modelo* é uma representação gráfica do modelo (estilo fluxograma) usando os símbolos dos blocos. A *declaração do modelo* é uma transcrição textual do diagrama, parecido com um programa de computador (escrito em linguagem de simulação), que é usado pelo motor de simulação. Há uma correspondência direta entre os blocos no diagrama do modelo e na declaração do modelo. A figura reffig:simandiagramamodelo1 apresenta o diagrama de um modelo simples, em que entidades chegam no sistema (com tempos entre chegada seguindo uma distribuição exponencial com média 20), alocam um recurso (machine), usam esse recurso por um tempo (que segue uma distribuição uniformemente distribuída entre 1 e 2), liberam o recurso e deixam o sistema. O código 2.3 mostra a declaração desse modelo em linguagem SIMAN, que descreve a estrutura do modelo, enquanto o código 2.4 descreve o plano de experimentos desse mesmo modelo, também como declarações em linguagem SIMAN. A simulação desse modelo gera um relatório de saída que pode ser visto na no código 2.5. Posteriormente a linguagem SIMAN foi incorporada ao simulador Arena, que passou a prover blocos de mais alto nível e a geração automática de código SIMAN a partir desses blocos mais complexos, facilitando o processo de modelagem.

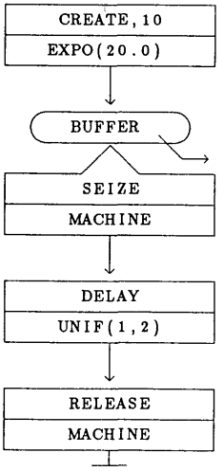

Figura 2.4: Exemplo de diagrama de modelo no SIMAN

```
1 $           CREATE,       1,SecondstoBaseTime(0.0),Entity
      1:SecondstoBaseTime(expo(20)):NEXT(6$);
2 6$           ASSIGN:       Create 1.NumberOut=Create 1.NumberOut + 1
```

```
;    Model statements for module: Seize 1
0$          QUEUE,        Seize 1.Queue;
SEIZE,       2,Other:     MACHINE,1:NEXT(10$);
10$         DELAY:        0.0,,VA:NEXT(2$);
;    Model statements for module: Delay 1
2$          DELAY:        UNIF(1,2),,Other:NEXT(3$);
;    Model statements for module: Release 1
3$          RELEASE:      MACHINE,1:NEXT(4$);
;    Model statements for module: Dispose 1
4$          ASSIGN:       Dispose 1.NumberOut=Dispose 1.NumberOut + 1;
11$         DISPOSE:      Yes;
```

Listing 2.3: Exemplo de código da estrutura de modelo com SIMAN

```
PROJECT,     "Unnamed Project","-",,,Yes,Yes,Yes,Yes,Yes,Yes,Yes,Yes,No,No;
VARIABLES:   Dispose 1.NumberOut, CLEAR(Statistics), CATEGORY("Exclude"):
Create 1.NumberOut,CLEAR(Statistics),CATEGORY("Exclude");
QUEUES:      Seize 1.Queue,FIFO,,AUTOSTATS(Yes,,);
RESOURCES:   MACHINE, Capacity(1),,, COST(0.0,0.0,0.0), CATEGORY(Resources),,
     AUTOSTATS(Yes,,);
REPLICATE,   30 ,, SecondsToBaseTime(3600), Yes,Yes,,,, 24, Seconds, No, No,,,
     Yes;
ENTITIES:    Entity 1, Picture.Report, 0.0,0.0,0.0,0.0,0.0,0.0,
     AUTOSTATS(Yes,,);
```

Listing 2.4: Exemplo de código do plano de experimento com SIMAN

```
SIMAN Simulation Results
- - License: STUDENT

Output Summary for 30 Replications
Project: Unnamed Project       Run execution date : 3/ 9/2020
Analyst:                       Model revision date: 3/ 9/2020

OUTPUTS
Identifier             Average   Half-width Minimum  Maximum # Replications
------------------------------------------------------------------------------

Entity 1.NumberIn       183.23    5.4430    151.00    213.00    30
Entity 1.NumberOut      183.23    5.4430    151.00    213.00    30
MACHINE.NumberSeized    183.23    5.4430    151.00    213.00    30
MACHINE.Utilization     .07638    .00226    .06307    .08949    30
MACHINE.BusyCost        .00000    .00000    .00000    .00000    30
MACHINE.IdleCost        .00000    .00000    .00000    .00000    30
MACHINE.UsageCost       .00000    .00000    .00000    .00000    30
All Entities.VACost     .00000    .00000    .00000    .00000    30
All Entities.NVACost    .00000    .00000    .00000    .00000    30
All Entities.WaitCost   .00000    .00000    .00000    .00000    30
All Entities.TranCost   .00000    .00000    .00000    .00000    30
```

```
All Entities.OtherCost  .00000    .00000    .00000    .00000    30
All Entities.TotalCost  .00000    .00000    .00000    .00000    30
All Resources.BusyCost  .00000    .00000    .00000    .00000    30
All Resources.IdleCost  .00000    .00000    .00000    .00000    30
All Resources.UsageCost .00000    .00000    .00000    .00000    30
All Resources.TotalCost .00000    .00000    .00000    .00000    30
System.TotalCost        .00000    .00000    .00000    .00000    30
System.NumberOut        183.23    5.4430    151.00    213.00    30

Simulation run time: 0.08 minutes.
Simulation run complete.
```

Listing 2.5: Exemplo de relatório de saída gerado pelo SIMAN

2.2.3 GenESyS

GenESyS (GENeric and Expasible SYstem Simulator) é uma biblioteca de simulação (e um simulador) desenvolvido pelo professor Rafael Cancian em suas atividades de ensino e pesquisa em modelagem e simulação de sistemas, e foi inicialmente baseado em SIMAN e em Arena, com foco em modelagem discreta e em ferramentas associadas, como simulação paralela e distribuída e geração automática de cenários de simulação para projeto fatorial de experimentos. O desenvolvimento do GenESys (então conhecido como SSD –Sistema de Simulação Discreta–) iniciou em 2004 e prosseguiu até 2007, quando foi encerrado. A partir de 2019 seu desenvolvimento foi recomeçado em nova linguagem de programação (C++) e com nova estrutura, visando também a modelagem de sistemas contínuos e diferentes técnicas de simulação, incluindo sistemas de equações diferenciais, volumes finitos, processos estocásticos, e autômatos celulares, além da simulação orientada a eventos que era a base da simulação dos modelos discretos que já suportava. GenESys visa suportar a simulação de modelos principalmente nas áreas de engenharias (de automação, mecânica, bioquímica, mecânica e ambiental) e de computação. Uma das vantagens do GenESys é ser de código aberto e organizado utilizando técnicas de engenharia de software que facilitam sua expansibilidade e seu uso pedagógico.

Em GenESyS modelos podem ser descritos de três formas diferentes: (1) diagrama do modelo, (2) declaração do modelo, ou (3) programação do modelo. O diagrama do modelo é análogo ao SIMAN e Arena, e permite a criação do modelo através de interface gráfica (ainda em desenvolvimento). A declaração do modelo utiliza uma linguagem semelhante ao SIMAN que pode ler entendida pelo motor de simulação do GenESyS. O código 2.6 apresenta um exemplo de modelo na linguagem GenESyS, que é o mesmo modelo apresentado em SIMAN no código 2.3. Por ser uma biblioteca de simulação, pode-se criar modelos e realizar a simulação diretamente em linguagem de programação C++, bastando incluir os cabeçalhos

das classes disponibilizadas. O código 2.7 apresenta o código de um programa que instancia o simulador, cria um modelo de simulação (o mesmo apresentado em forma de declaração anteriormente em GenESyS e em SIMAN), simula o modelo e mostra os relatórios de saída, que é apresentado na listagem 2.8.

```
0 SimulatorInfo version="19.10 (Halloween19)"
0 ModelInfo name="Model 1" replicationLength=3600.000000
    numberOfReplications=30
1 EntityType name="Any Entity"
2 Resource name="Machine" capacity=1
3 Queue name="Queue_Machine" orderRule=1
4 Create entityTypename="Any Entity" name="Create 1" nextId0=5
    timeBetweenCreations="expo(20)"
5 Seize name="Seize 1" queueName="Queue_Machine" resourceType=1
    resourceName="Machine" nextId0=6
6 Delay name="Delay 1" delayExpression="unif(1,2)" nextId0=7
7 Release name="Release 1" queueName="Queue_Machine" resourceName="Machine"
    nextId0=8 rule=4
8 Dispose name="Dispose 1"
```

Listing 2.6: Exemplo de código com linguagem de simulação GenESys

```
#include "Simulator.h"
#include "Create.h"
#include "Resource.h"
...
int main(int argc, char** argv) {
   Simulator* simulator = new Simulator();
   Model* model = new Model(simulator);
   ModelInfo* infos = model->infos();
   infos->setReplicationLength(3600);
   infos->setNumberOfReplications(30);
   EntityType* entityType1 = new EntityType(model, "Any_Entity");
   Create* create1 = new Create(model);
   create1->setEntityType(entityType1);
   create1->setTimeBetweenCreationsExpression("expo(20)");
   Resource* machine1 = new Resource(model, "Machine");
   Queue* queueSeize1 = new Queue(model, "Queue_Machine");
   Seize* seize1 = new Seize(model);
   seize1->setResource(machine1);
   seize1->setQueue(queueSeize1);
   Delay* delay1 = new Delay(model);
   delay1->setDelayExpression("unif(1,2)");
   Release* release1 = new Release(model);
   release1->setResource(machine1);
   Dispose* dispose1 = new Dispose(model);
   create1->nextComponents()->insert(seize1);
   seize1->nextComponents()->insert(delay1);
   delay1->nextComponents()->insert(release1);
```

```
28    release1->nextComponents()->insert(dispose1);
29    simulator->models()->insert(model);
30    model->simulation()->start();
31    model->simulation()->showReports();
32    simulator->~Simulator();
33 }
```

Listing 2.7: Exemplo de código com biblioteca de simulação GenESyS

```
1  GenESyS – GENeric and Expansible SYstem Simulator
2  Number of Replications : 30
3  Replication Length: 3600.000000 second
4   Begin of Report for Simulation (based on 30 replications )
5  |  |  Statistics for Create:
6  |  |  |  Create_1:
7  |  |  |  |  name                                    elems      min          max          average
       variance      stddev      varCoef     confInterv   confLevel
8  |  |  |  |  Count number in  ..........  30  124.000000  127.000000  126.433333
       0.445556    0.667499    0.005279    0.238862    0.950000
9  |  |  Statistics for Dispose:
10 |  |  |  Dispose_1:
11 |  |  |  |  name                                    elems      min          max          average
       variance      stddev      varCoef     confInterv   confLevel
12 |  |  |  |  Dispose_1.Count_number_out  ..........  30  124.000000  127.000000  126.433333
       0.445556    0.667499    0.005279    0.238862    0.950000
13 |  |  Statistics for EntityType:
14 |  |  |  Customer:
15 |  |  |  |  name                                    elems      min          max          average
       variance      stddev      varCoef     confInterv   confLevel
16 |  |  |  |  Delay_1.Waiting_Time  ..........  30  1.422298    1.423485    1.422871
       0.000000    0.000583    0.000410    0.000209    0.950000
17 |  |  |  |  Customer.Total_Time  ..........  30  1756.404335 1786.517452 1758.619362
       28.254600   5.315506    0.003023    1.902129    0.950000
18 |  |  Statistics for Queue:
19 |  |  |  Queue_Machine_1:
20 |  |  |  |  name                                    elems      min          max          average
       variance      stddev      varCoef     confInterv   confLevel
21 |  |  |  |  Queue_Machine_1.Number_In_Queue..... 30  0.000000    0.000000    0.000000
       0.000000    0.000000    0.000000    0.000000    0.950000
22 |  |  |  |  Queue_Machine_1.Time_In_Queue ..... 30  0.000000    0.000000    0.000000
       0.000000    0.000000    0.000000    0.000000    0.950000
23 |  |  Statistics for Resource:
24 |  |  |  Machine_1:
25 |  |  |  |  name                                    elems      min          max          average
       variance      stddev      varCoef     confInterv   confLevel
26 |  |  |  |  Machine_1.Time_Seized  ..........  30  1.422298    1.423485    1.422871
       0.000000    0.000583    0.000410    0.000209    0.950000
27 |  |  |  |  Machine_1.Seizes  ..........  30  124.000000  127.000000  126.433333
       0.445556    0.667499    0.005279    0.238862    0.950000
28 |  |  |  |  Machine_1.Releases  ..........  30  124.000000  127.000000  126.433333
       0.445556    0.667499    0.005279    0.238862    0.950000
29 |  End of Report for Simulation
```

Listing 2.8: Exemplo de relatório de saída gerado pelo GenESys

2.3 Simuladores

Um simulador é um programa de computador que realiza simulações. Em geral, além do "motor de simulação" eles possuem também diversas ferramentas acessórias para entrada e tratamento de dados, depuração, análise dos resultados e outras tarefas comuns na simulação. No contexto desta obra estamos interessados principalmente em simuladores genéricos, gratuitos e de código aberto, embora outros simuladores sem essas características também são citados. Iniciamos apresentando uma breve lista de alguns simuladores de interesse e então as subseções apresentam alguns simuladores com um pouco mais de detalhes.

Simulink é um simulador de código fechado, integrada no software MATLAB (pago), que permite modelar, simular e analisar sistemas dinâmicos de múltiplos domínios. Neste sentido uma grande variedade de sistemas podem ser projetados e testados, nomeadamente sistemas de controle, sistemas de processamento de sinal, sistemas de comunicação, sistemas logísticos, entre muitos outros. Ao contrário do MATLAB que utiliza a linha de comandos, o Simulink usa uma interface gráfica em que os modelos são criados sob a forma de diagramas de blocos, facilitando a interação com o utilizador. Ainda assim, esta ferramenta pode ser interligada com o MATLAB, podendo inclusive os modelos ser desenvolvidos através da linha de comandos. O Simulink inclui uma biblioteca de blocos pré-definidos, mas o utilizador pode igualmente criar os seus próprios blocos, além de possuir diversos "add-ons" para automatização de geração de código para sistemas embarcados, sistemas de tempo-real, sistemas digitais e outros.

Stateflow foi desenvolvido pela MathWorks e é uma ferramenta de lógica de controle usada para modelar sistemas reativos por meio de máquinas de estado e fluxogramas dentro de um modelo Simulink. O fluxo de estado usa uma variante da notação de máquina de estado finito, permitindo a representação da hierarquia, paralelismo e história dentro de um gráfico de estados. Stateflow também fornece tabelas de transição de estado e tabelas-verdade. O fluxo de estado é geralmente usado para especificar o controlador discreto no modelo de um sistema híbrido, onde a dinâmica contínua (ou seja, o comportamento da planta e do ambiente) é especificada usando o Simulink. Um número de ferramentas do MathWorks e de terceiros pode ser usado com o Stateflow para validar o design e gerar código.

Wolfram SystemModeler foi desenvolvido pela Wolfram MathCore e é uma plataforma para modelagem e simulação, com base na linguagem Modelica. Ele fornece um ambiente interativo de modelagem e simulação gráfica e um conjunto personalizável de bibliotecas de componentes. Os recursos do Wolfram SystemModeler incluem: Executa seus próprios cálculos simbólicos e numéricos, acessando as equações completas do modelo e os resultados da simulação de seus modelos;

Componentes reutilizáveis, permitindo a exploração rápida de projetos e cenários alternativos; Análises de frequência, sensibilidade e confiabilidade; Integração com o *Mathematica* para análise e documentação de modelos e simulações; Modelagem de domínios múltiplos, incluindo: mecânica, elétrica, hidráulica, engenharia de controle e biologia de sistemas.

2.3.1 Scilab-Scicos-Xcos

O Scilab é um software científico para computação numérica semelhante ao MATLAB que fornece um poderoso ambiente computacional aberto para aplicações científicas. Desenvolvido desde 1990 pelos pesquisadores do INRIA (Institut National de Recherche en Informatique et en Automatique) e do ENPC (École Nationale des Ponts et Chaussées), então pelo Consorcio Scilab desde Maio de 2003, Scilab é agora mantido e desenvolvido pelo Scilab Enterprises desde Julho de 2012. Distribuído gratuitamente via Internet desde 1994, o Scilab é atualmente usado em diversos ambientes industriais e educacionais pelo mundo. []

Scilab possui uma linguagem de programação de alto nível, orientada à análise numérica, baseada na sintaxe do Matlab. A linguagem provê um ambiente para interpretação, com diversas ferramentas numéricas. Scilab inclui centenas de funções matemáticas, sofisticadas estruturas de dados, e um grande número de ferramentas estão disponíveis, incluindo ferramentas para simulação, com resolvedor de sistemas de equações diferenciais explícitos e implícitos, e o Scicos/Xcos, um modelador gráfico e simulador de sistemas dinâmicos híbridos.

Scicos/Xcos é uma ferramenta Scilab dedicada à modelagem e simulação de sistemas híbridos dinâmicos, incluindo modelos contínuos e discretos. Também permite a simulação de sistemas governados por equações explícitas (simulação causal) e equações implícitas (simulação acausal). Os usuários podem criar diagramas de blocos para modelar e simular a dinâmica de sistemas dinâmicos híbridos e compilar esses modelos em código executável. As aplicações incluem processamento de sinais, controle de sistemas, sistemas de filas e o estudo de sistemas físicos, elétricos e biológicos. Xcos inclui um editor gráfico que permite representar facilmente modelos como diagramas de blocos, conectando os blocos uns aos outros. Cada bloco representa uma função básica predefinida ou definida pelo usuário [**http://www.scicos.org/**]. Ambos utilizam a linguagem de simulação Modelica, gerada automaticamente a partir dos modelos construídos graficamente.

Instalação

Scicos é distribuído com o pacote de software científico ScicosLab e está dis-

ponível para Windows (10, 8, 7, Vista, XP) e para Mac OS X. Ele pode ser baixado a partir do endereço `http://www.scicos.org/downloads.html`. XCos também é distribuído com o SciLab e está disponível para os mesmos sistemas operacionais e também para linux, e pode ser baixado a partir do endereço `https://www.scilab.org/download/6.1.0`. A instalação de ambos é muito simples, sendo que o Scicos possui um instalador gráfico (estilo "next, next, finish") e o arquivo baixado do XCos já é a estrutura de pastas do SciLab e seus executáveis estão na pasta `bin`. A partir da versão 5.x Scilab faz um uso avançado do JOGL (o Java Binding para OpenGL) e utiliza o Java2D OpenGL Pipeline, mais especificamente o buffer interno das placas gráficas chamado pbuffer. Isso pode causar algum problema na visuação dos gráficos gerados como resultados das simulações e se ocorrer será necessário pesquisar por soluções para seu driver de vídeo. Uma solução comum para o problema das janelas de resultados não mostrarem gráficos é executar o Scilab usando o seguinte comando (estando na pasta onde ele foi instalado): `LIBGL_ALWAYS_SOFTWARE=1 bin/scilab`.

Primeiro exemplo

Vamos iniciar noos primeiro exemplo no Scicos. Execute o Scicos a partir do editor de comandos do SciLab (com o comando `scicos;`) ou a partir do menu "`aplicativos`"e então "`xcos`". A tela inicial deve ser como a que aparece na figura 2.5, com duas janelas: o navegador de paletas (à esquerda na figura) e o editor de diagramas de modelos (à direita na figura). No navegador de paletas podemos escolher entre muitos blocos/componentes de modelo de diferentes paletas, as quais as mais relevantes a esta obra são: Sistemas de tempo contínuo, Sistemas de tempo discreto, Elétrica, Receptores (*Sinks*) e Fontes (*Sources*). Nesse primeiro exemplo vamos criar um modelo contínuo baseado num sistema de quações diferenciais que represente a disseminação de uma doença. Esse modelo se baseia no sistema de equações diferenciais abaixo.

$$\begin{aligned} \dot{x}_1(t) &= \alpha.x_1(t).x_2(t) \\ \dot{x}_2(t) &= \alpha.x_1(t)).x_2(t) - \beta.x_2(t) \\ \dot{x}_3(t) &= \beta.x_2(t) \end{aligned}$$

, onde x_1 é a fração de pessoas suscetíveis à doença, x_2 é a fração de pessoas infectadas pela doença e x_3 é a quantidade de pessoas recuperadas da doença (imunes). Assume-se que ninguém morre dessa doença, de modo que a fração total de pessoas é sempre $x_1 + x_2 + x_3 = 1$. Ainda, nesse sistema, α é a taxa de infecção e β é a taxa de recuperação da doença. Os valores iniciais para esse sistema de equações diferenciais são $x_1 = 0.99$, $x_2 = 0.01$ e $x_3 = 0$, e os parâmetros são $\alpha = 1$ e $\beta = 0.35$.

Esse modelo pode ser descrito no Scicos conforme ilustrado na figura 2.6. Os

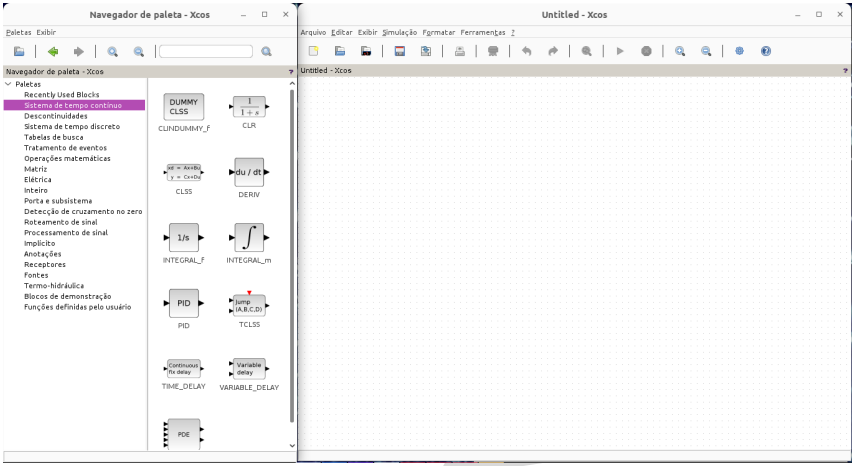

Figura 2.5: Tela inicial do simulador SciCos

blocos com sinal de integração são integradores (blocos `INTEGRAL_m`, da paleta "Sistemas de tempo contínuo"). Perceba que cada equação diferencial foi modelada com uma função integral, dada por $x(t) = \int_{t_0}^{t} \dot{x}(\tau)d\tau$, onde $\dot{x}$ é o sinal de entrada. Como a saída é a integral da entrada, então a qualquer instante a entrada é a derivada da saída, ou seja, $\dot{x}(t) = \frac{d}{dt}x(t)$ (integrando a derivada $\dot{x}$ obtem-se a função x).

Além do bloco `INTEGRAL_m` esse modelo utiliza os blocos `GAIN_f` (da paleta "Operações matemáticas) para representar os parâmetros α e β, `PROD_f` e `SUMMATION`(da paleta "Operações matemáticas) para multiplicar e somar as funções, `MUX` (da paleta "Roteamento de sinal") para agrupar valores num vetor, `CSCOPE` (da paleta "Receptores") para plotar o valor das equações diferenciais ao longo do tempo, e `CLOCK_c` para gerar eventos para atualização do gráfico.

Para construir esse modelo, basta acessar a paleta que contém o bloco desejado (na janela de "Navegador de paleta"), selecionar e arrastar o componente para a área de edição da janela de editor de diagrama do modelo. Depois de posicionar os blocow é necessário conectá-los (como na figura 2.6). Para isso, pressione o botão do mouse sobre a saída de um bloco e segure o botão enquanto arrasta o mouse até a entrada do bloco destino. Para criar uma "intersecção", conectando mais de um bloco à saída, selecione uma conexão existente (onde a interseção será inserida) e proceda como descrito anteriormente para conectar o novo bloco destino. Eventuais parâmetros dos blocos podem ser editados após um duplo clique no bloco ou clicando

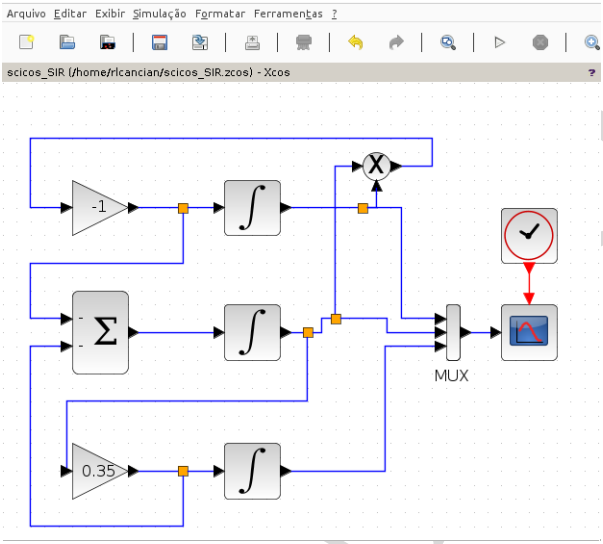

Figura 2.6: Modelo de disseminação de doença

com o botão direito e então selecionado "Parâmetros do bloco...", ou usando o atalho de techado "CTRL+B". Você precisará editar parâmetros dos blocos `GAIN_f` para definir os valores de α e β, e também do bloco `CSCOPE` para definir os eixos do gráfico. A duração da simulação pode ser definida pelo menu "`Simulação`"e então "`Configuração`". Para executar a simulação, acesse o menu "`Simulação`"e então "`Iniciar`".

O resultado da simulação desse modelo é apresentado na figura 2.7. A curva preta representa x_1, a fração de pessoas suscetíveis, que diminui à medida em que pessoas vão se infectando e ficando imunes. A curva verde representa x_2, a fração de pessoas infectadas, que aumenta e depois diminui, e a curva vermelha representa x_3, a fração de pessoas recuperadas (imunes), que aumenta à medida que pessoas infectadas vão se recuperando.

Você também pode observar vários modelos de demonstração no Scicos, se quiser. Para isso, na janela de edição de diagramas de modelo, acesse o menu "`?`"(ajuda) e então "`Xcos Demonstrations`". Na janela que aparece, selecione "`Xcos`"na lista da esquerda e então a demonstração quiser, na lista que aparecerá à direita.

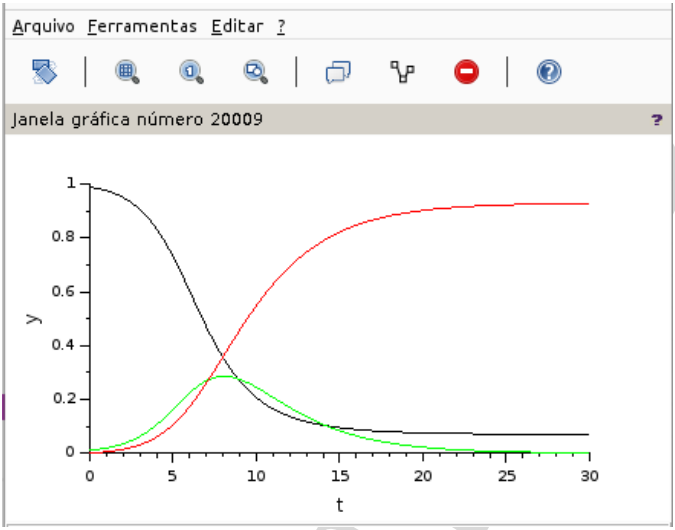

Figura 2.7: Resultado da simulação do modelo de disseminação de doença

2.3.2 Ptolemy II

Ptolemy II [**PtolemyProject**] é um software de código aberto que suporta experimentação com design orientado a ator. Atores são componentes de software que são executados simultaneamente e se comunicam através de mensagens enviadas por portas interconectadas. Um modelo é uma interconexão hierárquica de atores. Em Ptolemy II, a semântica de um modelo não é determinada pela estrutura, mas por um componente de software no modelo chamado *diretor*, que implementa um modelo de computação. O Projeto Ptolemy desenvolveu diretores que oferecem suporte a redes de processos (PN), eventos discretos (DE), fluxo de dados (SDF), modelos síncronos/-reativos (SR), baseados em *rendevouz*, visualização 3D e modelos de tempo contínuo. Cada nível da hierarquia em um modelo pode ter seu próprio diretor, e diretores distintos podem ser compostos hierarquicamente. Uma ênfase principal do projeto tem sido a compreensão das combinações heterogêneas de modelos de computação realizados por esses diretores. Os diretores podem ser combinados hierarquicamente com máquinas de estado para criar modelos modais. Uma combinação hierárquica de modelos de tempo contínuo com máquinas de estados produz sistemas híbridos (muito úteis para sistemas cyber-físicos); uma combinação de síncrona/reativa com máquinas de estado produz StateCharts.

Ptolemy II está em desenvolvimento desde 1996 e é um sucessor do Ptolemy

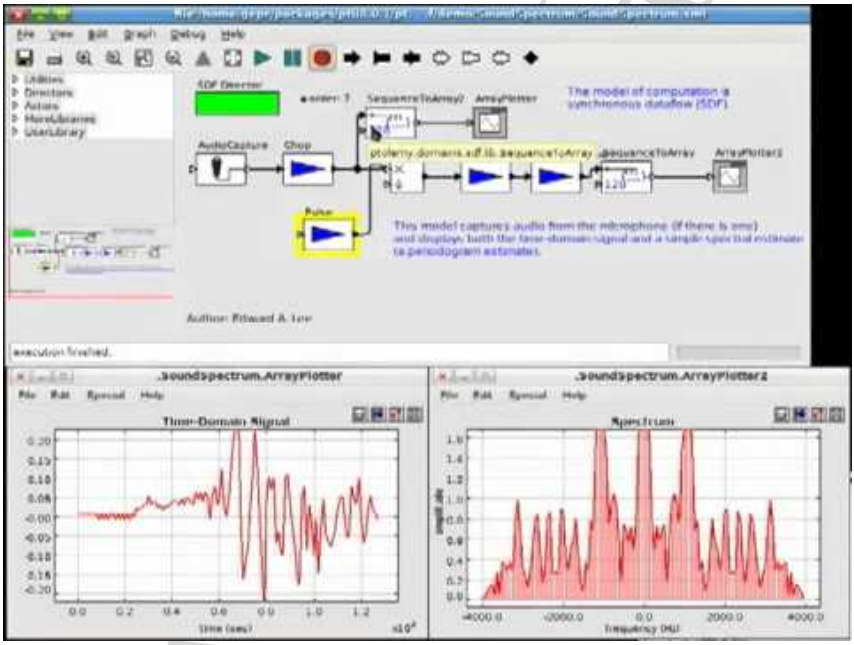

Figura 2.8: Exemplo de modelo no Ptolemy II

Classic, desenvolvido desde 1990. O núcleo do Ptolemy II é uma coleção de classes e pacotes Java, em camadas para fornecer recursos cada vez mais específicos. O kernel suporta uma sintaxe abstrata, uma estrutura hierárquica de entidades com portas e interconexões. Um editor gráfico chamado Vergil suporta a edição visual dessa sintaxe abstrata. Uma sintaxe concreta XML chamada MoML fornece um formato de arquivo persistente para os modelos. Várias ferramentas especializadas foram criadas a partir dessa estrutura, incluindo HyVisual (para modelagem de sistemas híbridos), Kepler (para fluxos de trabalho científicos), VisualSense (para modelagem e simulação de redes sem fio), Viptos (para design de rede de sensores) e alguns produtos comerciais. As principais partes da infraestrutura incluem uma semântica abstrata de ator, que permite a interoperabilidade de modelos distintos de computação com uma semântica bem definida; um modelo de tempo (especificamente, tempo super denso, que permite a interação de dinâmica contínua e lógica imperativa); e um sofisticado sistema de tipos que suporta verificação de tipos, inferência de tipos e polimorfismo. O sistema de tipos foi recentemente estendido para suportar ontologias definidas pelo usuário [6]. Vários experimentos com síntese de código-fonte e abstrações para verificação estão incluídos no projeto. [**PtolemyApproach**]

Ptolemy II é baseado numa classe de modelagem denominada **orientada a atores**, ou simplesmente modelagem por atores. Atores são componentes que executam concorrentemente e compartilham dados com outros componentes pelo envio de mensagens via ports. O conjunto de todas as mensagens comunicadas via um port é chamado de sinal. O bloco `Director` especifica o domínio do modelo e, assim, o **modelo de computação**. Um modelo hierárquico de atores consiste de um ator composto de *top-level* e atores que são submodelos, cada um dos qual sendo também um ator composto, como aprresenta a figura 2.9. Modelos complexos são tipicamente hierárquicos. Um bom modelo de um sistema complexo provê modelos com visão simplificada, interfaces simples e baixo acoplamento, de modo a facilitar o entendimento e análise do modelo, bem como prover reusabilidade aos componentes.

No Ptolemy II uma implementação de um modelo de computação é chamado de domínio. Todos os domínios no Ptolemy II são garantidamente determinísticos a não ser quando o modelo explicitamente especifica comportamento não-determinístico. Um domínio é dito determinístico quando os sinais enviados entre atores não dependem de decisões arbitrárias de escalonamento, apenas da concorrência do modelo. Garantir determinismo não é nem um pouco trivial em modelos de computação concorrentes, bem como prover mecanismos confiáveis de não determinismo. A explicação de cada tipo de domínio pode ser encontrada na documentação da linguagem. [**Claudius Ptolemaeus**, **Editor**, **System Design**, **Modeling**, **and Simulation using Ptolemy II**, **Ptolemy.org**, **2014.http://ptolemy.org/systems.**].

Instalação

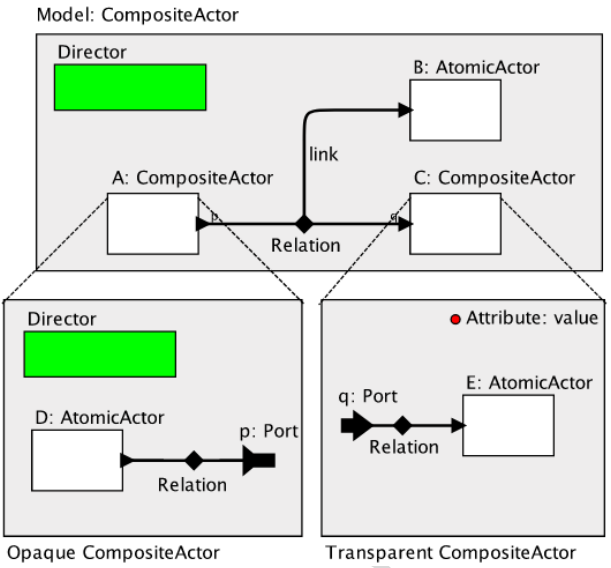

Figura 2.9: Modelagem baseada em atores e hierarquia de modelos no Ptolemy II

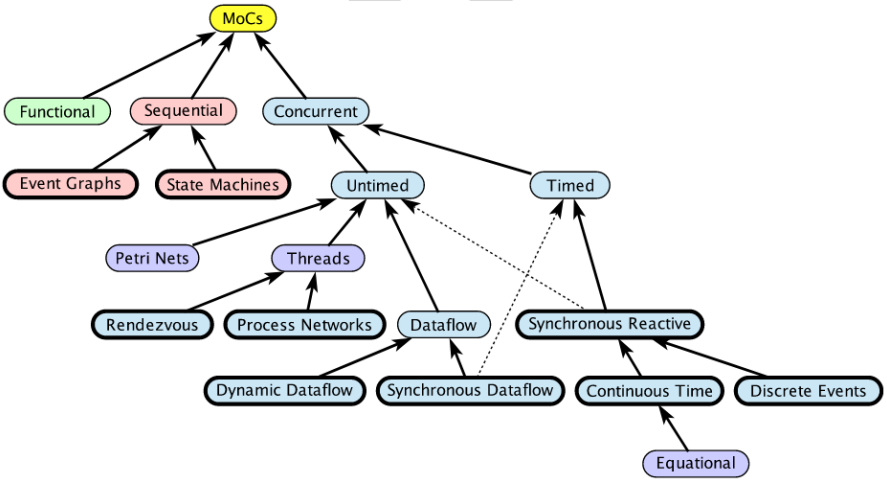

Figura 2.10: Modelos de computação e domínios no Ptolemy II

O simulador pode ser obtido a partir de sua página web em `https://ptolemy.berkeley.edu/ptolemyII/index.htm`, pelo menu *Download*. No momento da escrita deste texto a versão mais recente é a *Ptolemy II 11.0.1* de 19 de junho de 2018. O arquivo a ser obtido é o *ptII11.0.1.src.tar.gz*. Verifique se você possui todas as dependências de pacotes (ex: `npm`) e requisitos exigidos e faça o download do arquivo. Depois do download concluído e do arquivo descompactado, mude (`cd`) para a pasta inicial e então crie a variável de ambiente `PTII` apontando para a pasta de instalação: `export PTII=‘pwd‘`. Execute `./configure` e então instale o simulador `make fast install`, o que levará vários minutos. Se algo não funcionar, leia a documentação e resolva o problema. Depois de instalado, invoque o simulador Ptolemy II: `$PTII/bin/vergil`. A tela inicial deve ser como a apresentada na figura 2.11.

Figura 2.11: Tela de abertura do Ptolemy II

Primeiro exemplo

Vamos iniciar executando um exemplo pronto do Ptolemy II. Para isso, na tela inicial, selecione *Tour of Ptolemy II*. Na página que abre, selecione a categoria *Concurrent Models of Computation*. Nesa categoria, vamos usar um modelo contínuo (*Continuous-Time Modeling*): Lorenz, que é o famoso atrator de Lorenz, um sistema não linear com feedback que exige comportamento caótico. Ao selecionar Lorenz, seu modelo é exibido, como apresentado na figura 2.12. Esse diagrama de blocos é uma representação de um conjunto de equações diferenciais ordinárias não lineares. Os blocos com sinal de integração são integradores, e a saída do modelo é dada por

$$x(t) = \int_{t_0}^{t} \dot{x}(\tau) d\tau$$

onde $x(t_0)$ é o estado inicial do integrador, t_0 é o tempo inicial do modelo, e $\dot{x}$ é o sinal de entrada. Note que já que a saída é a integral da entrada, então a qualquer instante a entrada é a derivada da saída, ou seja,

$$\dot{x}(t) = \frac{d}{dt}x(t)$$

Então, o sistema descreve tanto uma equação integral quanto uma equação diferencial, dependendo de qual dessas formas você use.

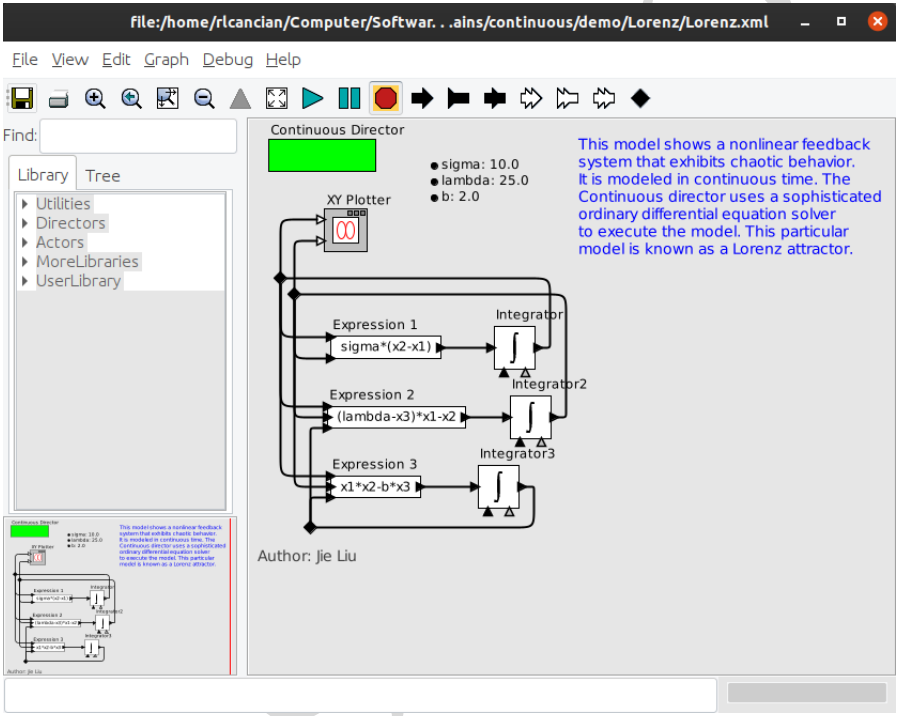

Figura 2.12: Modelo do atrator de Lorenz

Suponha que a saída do integrador no topo do diagrama da figura 2.12 seja x_1, que a saída do integrador do meio seja x_2 e que a saída do integrador de baixo seja x_3, então as equações diferenciais descritas nesse modelo são:

$$\begin{aligned}
\dot{x}_1(t) &= \sigma(x_2(t) - x_1(t)) \\
\dot{x}_2(t) &= (\lambda - x_3(t))x_1(t) - x_2(t) \\
\dot{x}_3(t) &= x_1(t)x_2(t) - bx_3(t)
\end{aligned}$$

Para cada equação, a expressão à direita é implementada por um ator `Expression`, cujo ícone mostra a expressão. Cada expressão refere-se a parâmetros (como lambda para λ e sigma para σ) e portas de entrada do ator (como x1 para x_1 e x2 para x_2). Os nomes das portas de entrada não são mostrado no diagrama, mas se você

permanecer com o cursor do mouse sobre eles, o nome aparecerá em uma "dica" da ferramenta. A expressão em cada ator `Expression` pode ser editada clicando duas vezes no ator, e os valores dos parâmetros podem ser editados clicando duas vezes nos parâmetros, que são mostrados ao lado de marcadores à direita. Os integradores também têm valores iniciais, que você pode examinar e alterar clicando duas vezes no ícone do integrador correspondente. Eles definem os valores iniciais de x_1, x_2 e x_3, respectivamente. Para este exemplo, todos os três estão definidos como 1.0.

O *Solver* de Tempo Contínuo (CT – *Continuos-Time*), mostrado no canto superior esquerdo, gerencia uma simulação do modelo. Ele contém um sofisticado solucionador/resolvedor de ODE (*Ordinary Differential Equation*) e, para usá-lo efetivamente, você precisará entender alguns de seus parâmetros. Os parâmetros são acessados clicando duas vezes na caixa do solucionador. Os mais simples desses parâmetros são o *startTime* e o *stopTime*, que definem a região da linha do tempo na qual uma simulação será executada. Outros parâmetros importantes são *initStepSize*, *maxStepSize* e *maxIterations*. Para executar o modelo, você pode clicar no botão de execução na barra de ferramentas (com um ícone de triângulo tipo "play") ou você pode abrir a janela "Executar" no menu "Exibir". No primeiro caso, o modelo é executado e o os resultados são plotados em sua própria janela, como mostra a figura 2.13. O que é plotado é $x_1(t)$ num eixo e $x_2(t)$ noutro eixo, para valores de t entre *startTime* e *stopTime*.

Depois de compreender esse modelo, retorne à na tela inicial, selecione *Tour of Ptolemy II* e explore outros modelos de seu interesse. Sugerimos os seguintes exemplos:

- *Thermostat* em *Hybrid systems* (um modelo híbrido: contínuo+discreto), pois mostra um modelo contínuo (equações diferenciais) sendo controlado por um modelo discreto (uma máquina de estados finitos), num exemplo simples de sistema cyber-físico.
- *QueueAndServer* em *Discrete-Event Modeling*, que mostra um exemplo simples de um servidor com uma fila de espera, semelhante ao exemplo apresentado nas seções anteriores sobre SIMAN e GenESyS.
- *HiddenMarkovModelAnalysis* em *Statistical Models*, que mostra uma cadeia de markov com três estados e distribuição normal.

2.3.3 OpenModelica

OpenModelica é um software de código aberto que tem como um de seus objetivos de curto prazo desenvolver um ambiente computacional interativo eficiente para a linguagem Modelica, bem como uma implementação bastante completa da linguagem. Acontece que, com o suporte de ferramentas e bibliotecas apropriadas, o Modelica

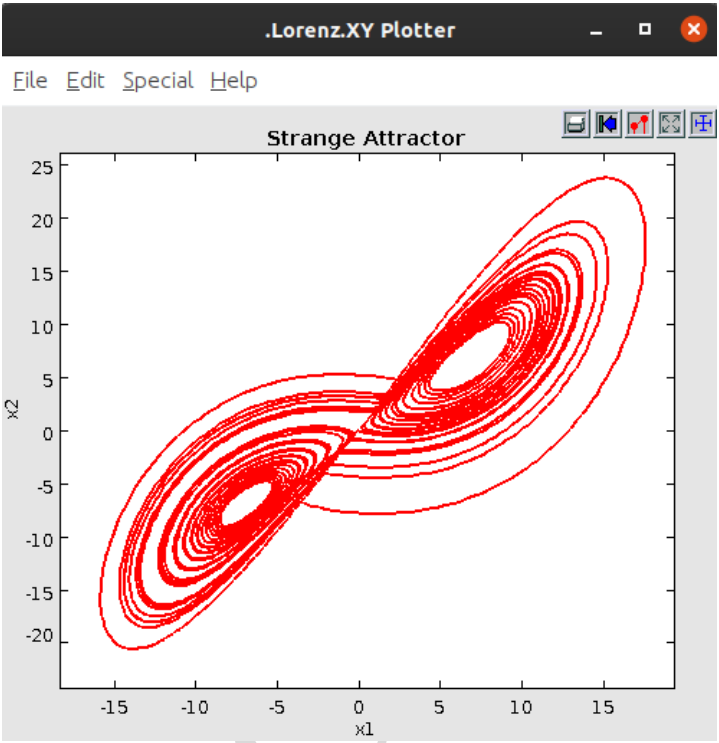

Figura 2.13: Resultado da simulação do modelo do atrator de Lorenz

é muito adequado como uma linguagem computacional para desenvolvimento e execução de algoritmos numéricos de baixo e alto nível, por exemplo para o projeto de sistemas de controle, resolução de sistemas de equações não lineares ou para desenvolver algoritmos de otimização aplicados a aplicações complexas

O objetivo do OpenModelica a longo prazo é ter uma implementação de referência completa da linguagem Modelica, incluindo simulação de modelos baseados em equações e facilidades adicionais na programação ambiente, bem como instalações convenientes para pesquisa e experimentação em design de linguagem ou outras atividades de pesquisa.No entanto, seu objetivo não é atingir o nível de desempenho e qualidade fornecido pelos atuais ambientes comerciais da Modelica que podem lidar com modelos grandes que exigem análise e otimização avançadas pelo compilador da Modelica.

Instalação

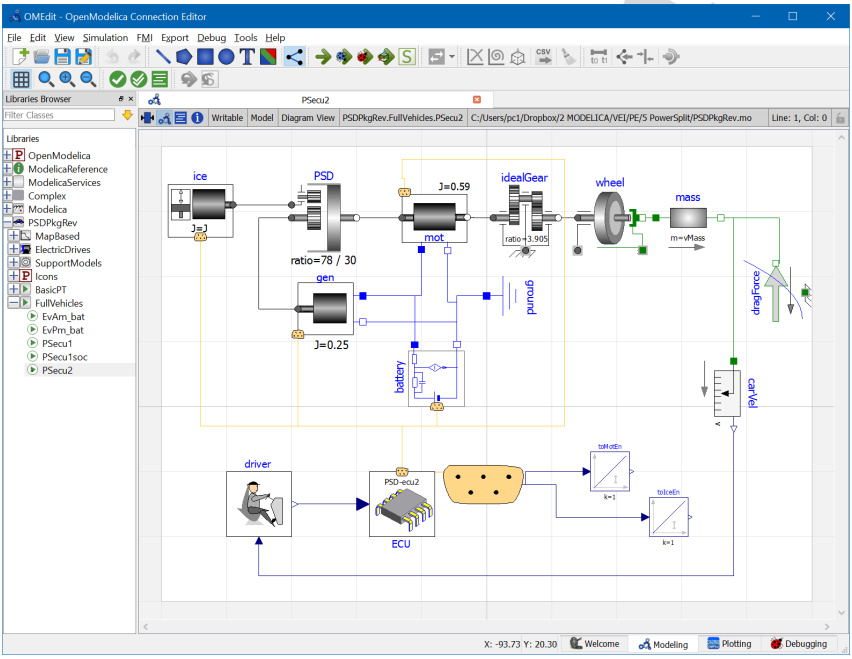

Figura 2.14: Exemplo de modelo no Openmodelica

Informações sobre a instalação do OpenModellica podem ser obtidas em sua página web oficial `https://www.openmodelica.org/`, no meu *Download*. No momento da escrita deste texto, a última versão oficial é a 1.14.1 e a última versão estável é a 1.15.0-dev. Dentre as diferentes opções de instalação, você precisará incluir algumas coisas no gerenciador de pacotes, executando

```
for deb in deb deb-src; do echo "$deb http://build.openmodelica.org/apt
    `lsb_release -cs` stable"; done | sudo tee
    /etc/apt/sources.list.d/openmodelica.list
```

incluir a chave GPG usada para assinar essas versões, executando

```
wget -q http://build.openmodelica.org/apt/openmodelica.asc -O- | sudo apt-key
    add -
apt-key fingerprint
```

e então instalar o OpenModelica e suas bibliotecas:

```
sudo apt update
sudo apt install openmodelica
sudo apt install omlib-.*
```

O principal executável instado é o OMEdit (*Connection Edition*), normalmente instalado em `/usr/bin`.

Primeiro Exemplo

#ToDo: Escolher exemplo e escrever esta seção.

2.3.4 Arena

Em 1982 foi lançada primeira versão da linguagem de simulação SIMAN, que foi desenvolvida pela Systems Modelyng Corporation (EUA), inspirada na linguagem GPSS usada em computadores de grande porte. A ideia era inovadora e foi a primeira linguagem específica de simulação destinada a IBM PC compatíveis. Em 1990, foi lançado o pacote CINEMA, que, integrado ao SIMAN permitia apresentar uma representação animada e em cores do funcionamento do sistema. Sendo inovadora mais uma vez, foi a primeira interface do tipo para simulação. Em 1993, SIMAN e CINEMA evoluiram e foram integrados em um ambiente único de simulação que unia e potencializava seus recursos, o software ARENA. A linguagem SIMAN, através do software ARENA, passou a ser representada em formato gráfico, tornando-se bastante intuitiva e agradável.

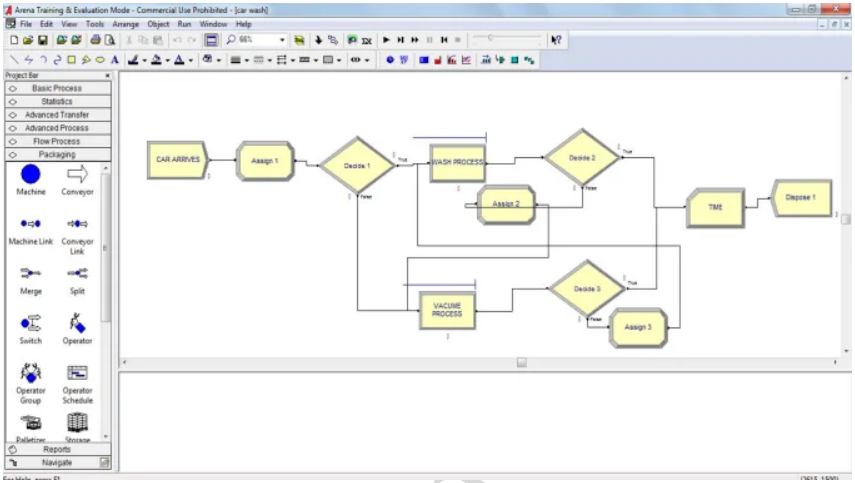

Figura 2.15: Exemplo de modelo no Arena

O software ARENA é um ambiente gráfico integrado de simulação, de código fechado e pago, que contém todos os recursos para modelagem de processos, desenho e animação, análise estatística e análise de resultados. É software de Simulação mais utilizado no mundo, com mais de 350.000 usuários. O software Arena permite a modelagem e simulação de diversos processos. Ele é muito utilizado para a análise de filas, de linhas de produção e também de processos industriais contínuos. Como qualquer software de simulação, ele permite prever o comportamento de algo que não existe no mundo real. Por exemplo, se uma indústria pretende fazer alguma mudança, a utilização do software Arena para modelar e simular a nova situação permite prever o comportamento futuro da usina, permitindo a validação ou mudança do projeto de modificação da indústria. Arena é compatível com os programas da Microsoft. Isso inclui a leitura e escrita de dados em Microsoft Office Access e Microsoft Office Excel.

Instalação

O Arena está disponível apenas para o sistemas operacional Windows. A versão de estudante do simulador Arena pode ser obtida a partir da página web `https://www.arenasimulation.com/`, seguindo o link *Download a Free Trial*. No momento em que este texto é escrito, a versão mais atual é a 15.0. Você precisará se registrar para poder fazer o downalod do simulador que possuirá restrições quanto ao tamanho do modelo que pode ser construído e quanto à quantidade de entidades simuladas simultaneamente. Após concluído o download, basta executar o instalador e seguir

as instruções fornecidas. Além do Arena podem ser instaladas outras ferramentas de apoio, como o Input Analyser, Output Analyser, OptQuest e Process Analyser. Recomenda-se a instalação de todas as ferramentas.

Primeiro Exemplo

Ao executar o software Arena, acesse o menu *"File" / "Open"* e então localize a pasta onde o Arena foi instalado e a subpasta *"Examples"*. Dentre os muitos exemplos de modelos disponíveis, escolha o *"Work in Process"*. Ao abrir esse exemplo você deve ver a tela como a apresentada na figura 2.16.

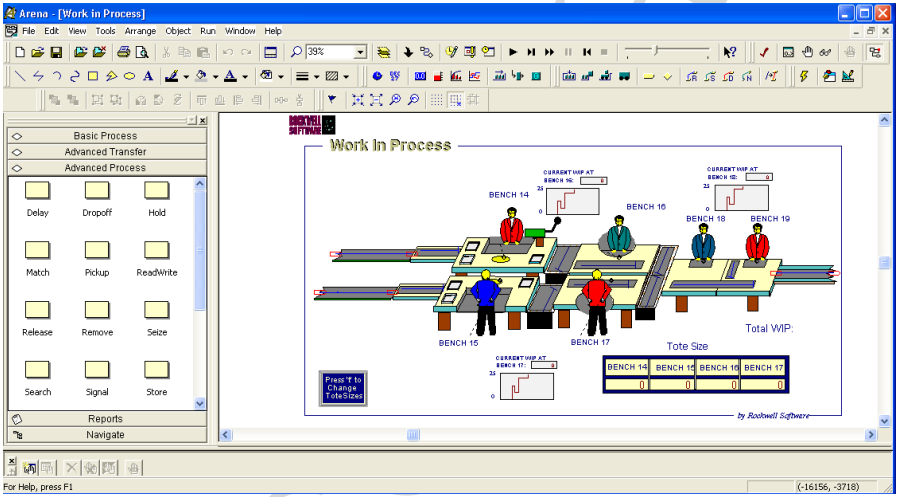

Figura 2.16: Modelo Work in Process aberto no Arena

Para executar o modelo basta pressionar o ícone "Go" na barra de ferramentas (um triângulo preto no estilo "play") ou acessar o menu "Run" e então "Go". Esse exemplo é uma simulação semi-interativa, e o usuário pode alterar alguns parâmetros quando a simulação inicia. Após confirmar o valor dos parâmetros a simulação prossegue, podendo ser acompanhada pela animação das figuras na tela. Ao término da simulação você poderá ver o relatório de saída, que contém todas as estatísticas geradas pelo modelo, como o apresentado na figura 2.17.

O modelo no Arena não é a animação, mas um diagrama de blocos orientado a processos, como no SIMAN. Na verdade, é SIMAN o que está por trás dos modelos visuais do Arena. Para visualizar o modelo, pressione a tecla "L", pois esse é o atalho que foi definido para visualização da lógica desse modelo. O modelo apresentado parece bem simples, mas é porque ele é hierárquico, e os blocos com a seta para baixo

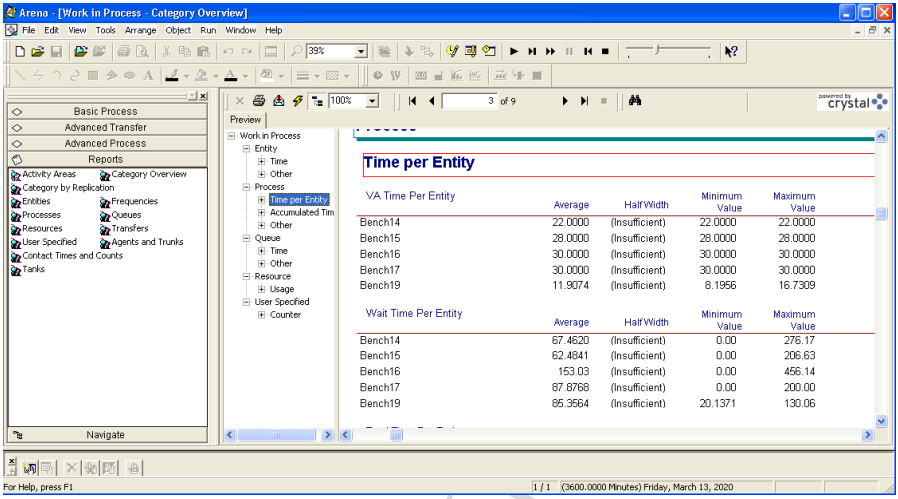

Figura 2.17: Relatório de saída do modelo Work in Process

na verdade são submodelos. Se você der um clique duplo sobre esses componentes (ou se clicar com o botão direito e selecionar *"Open Submodel"*), o sub-modelo será aberto, e você verá seu diagrama de blocos. Para voltar ao nível hierárquico superior, clique com o botão direito sobre qualquer lugar no fundo do modelo e selecione *"Close submodel"*. Se você der um clique duplo sobre qualquer componente do diagrama poderá ver seus parâmetros e ajustá-los. Há um botão de ajuda em cada um deles para que você explore o que eles fazem.

Depois de compreendido o modelo, recomendo acessar o menu *Help* e então selecionar a opção *"Arena Smarts File"*, que traz centenas de pequenos modelos desenvolvidos com o objetivo de ensinar aspectos e componentes específicos, sendo um bom guia para o aprendizado.

2.3.5 GenESyS

GenESyS é um software gratuito de código aberto inspirado no Arena e objetiva ser um simulador genérico e expansível (*Generic and Expansible System Simulator*), capaz de simular sistemas contínuos e discretos utilizando diferentes técnicas de modelagem. Ele foi criado pelo professor Rafael Cancian e está sendo desenvolvido na Universidade Federal de Santa Catarina em linguagem C++ desde 2019 e consiste num núcleo de simulação orientado a eventos (como o Arena), ferramentas de suporte estatístico, um conjunto de componentes (blocos) e pode ser acessado por console,

por uma interface gráfica ainda em desenvolvimento (ver figura 2.18) ou remotamente (socket IP, em desenvolvimento). Além de blocos análogos ao do software Arena, ele possui blocos que permitem a modelagem de sistemas contínuos por equações diferenciais e modelagem por cadeias de Markov e autômatos celulares.

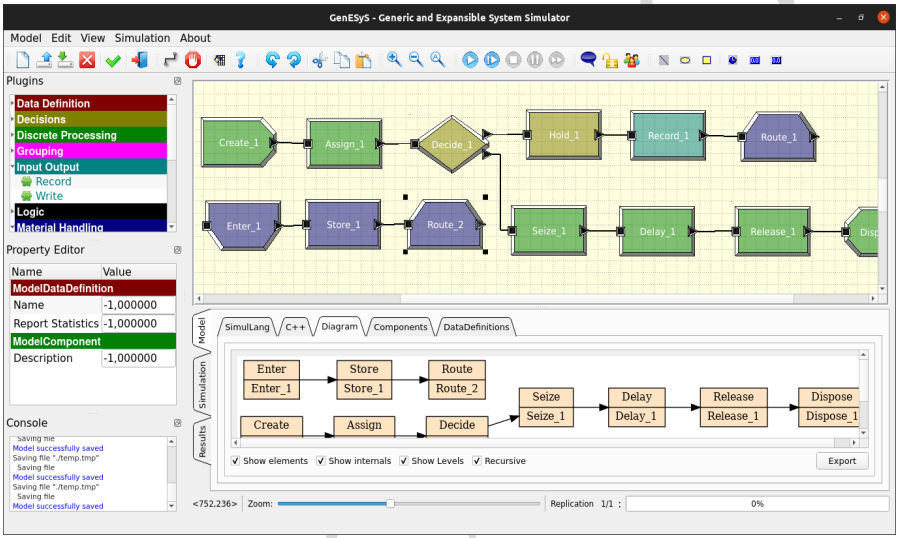

Figura 2.18: Exemplo da interface gráfica do GenESyS, ainda em desenvolvimento

A arquitetura geral do GenESyS é apresenta na figura 2.19, organizada em camadas. Nela, o principal componente é o `GenESyS Kernel`, responsável por fornecer todos os serviços de simulação de modelos. Ele deve ser ligado dinamicamente a um software qualquer para criar uma aplicação, como representado no componente `Applications`. Se a aplicação precisar, ela também pode ligar dinamicamente ferramentas de simulação (`Tools`) disponibilizadas pelo GenESyS. Por ser um simulador expansível, os comportamentos representados em modelo de simulação são `Plugins` que devem ser ligados dinamicamente ao `Kernel`, do mesmo modo que um `Parser`, já que plugins podem definir expressões que precisam ser avaliadas pelo parser. Na versão acadêmica as ligações dinâmicas estão desativadas e todos os componentes são compilados juntos, o que facilita a depuração e entendimento dos princípios de simulação.

As principais classes do `Kernel` do GenESyS associadas à simulação (classe `Simulator`) podem ser vistas na figura 2.20. O acesso a todas as demais classes se dá pela classe `Simulator`, que pode ser considerada a principal classe do GenESyS.

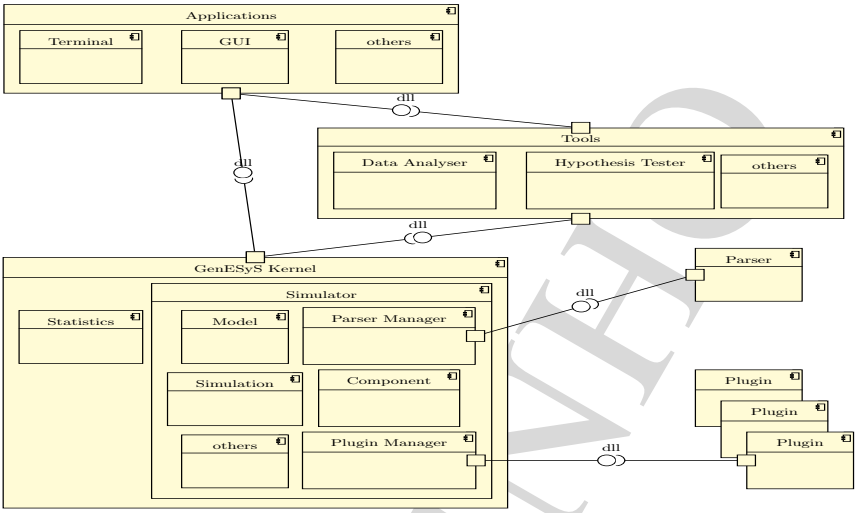

Figura 2.19: Arquitetura geral do GenESyS

Uma descrição geral e superficial das principais responsabilidades de cada classe apresentada nessa figura é dada a seguir:

- Simulator: Fornece acesso às classes associadas (`ParserManager`, `TraceManager`, etc).
- ParseManager: Conecta (carrega dinamicamente) um parser ao simulador.
- ExperimentManager: Gerencia experimentos de simulação, definindo cenários de simulação, parâmetros de entrada dos modelos e resultados a serem coletados em cada experimento.
- TraceManager: Gerencia os traces (ou logs) gerados pelo simulador e onde eles serão apresentado.
- PluginManager: Gerencia os plugins, permitindo inserir, excluir e encontrar plugins.
- PluginConnector: Define a forma como um plugin é inserido (carregado, conectado) ao simulador.
- Plugin: Permite criar um novo componente (ModelComponent) ou elemento (ModelElement) definido pelo plugin.
- ModelManager: Gerencia modelos de simulação, permitindo inserir, remover, carregar e salvar modelos.
- Model: Representa um modelo de simulação. Possivelmente a classe mais

importante do GenESys. Fornece acesso às suas classes associadas (ModelInfo, ModelSimulation, ComponentManager, ElementManager, etc).
- ModelInfo: Guarda informações básicas sobre o modelo.
- ModelSimulation: Realiza a simulação do modelo, permitindo iniciar, pausar, continuar, parar a simulação, definir os parâmetros de simulação e obter acesso aos relatórios de simulação.
- Event: Representa um evento a ser simulado, e contém o tipo de evento (um ModelComponent), a entidade que gera esse evento e o instante de ocorrência do evento.
- ComponentManager: Gerencia componentes do modelo de simulação, permitindo, inserir, editar, remover e encontrar componentes.
- ModelComponent: Representa um comportamento que pode ser simulado e vinculado a um tipo de evento específico. Um componente é criado a partir de um plugin específico. Possui métodos que descrevem o comportamento simulado, que permitem verificar a corretude de seus parâmetros, carregar e salvar uma instância do componentes, entre outros. Componentes são inseridos no modelo para formar um fluxo ou processo pelo qual entidades devem seguir.
- ElementManager: Gerencia elementos do modelo de simulação, permitindo, inserir, editar, remover e encontrar elementos.
- ModelElement:Representa informações ou estruturas mais básicas que um componente, e são acessados apenas pelos componentes do modelo, não fazendo parte diretamente do fluxo do modelo.
- OnEventManager: Gerencia a interceptação de eventos gerados pelo simulador, como o início e término de uma simulação, a pausa de uma replicação, entre outros. Permite que outras classes possam ser notificadas quando um evento específico acontece.
- SimulationEventHandler: Um tipo que define uma função que intercepta um evento de simulação.
- SimulationControl: Representa um parâmetro de entrada ou de configuração do modelo, que pode ser ajustado em diferentes cenários de execução.
- SimulationResponse: Representa uma saída ou resposta gerada pela simulação de um modelo, que pode ser obtida em diferentes cenários de execução.

Instalação

GenESys stá disponível no GitHub pelo endereço `https://github.com/rlcancian/Genesys-Simulator`. O GenESyS, em seu estado atual de desenvolvimento, não é instalado. Num sistema com Ubuntu basta executar os comandos abaixo (2.9) para instalar os pacotes de dependência e clonar o repositório do GenESyS (ou seja, fazer download de seu código-fonte). Esses comandos assumem que o sistema possui apenas o *Ubuntu Desktop 20.04.3 LTS* recém instalado.

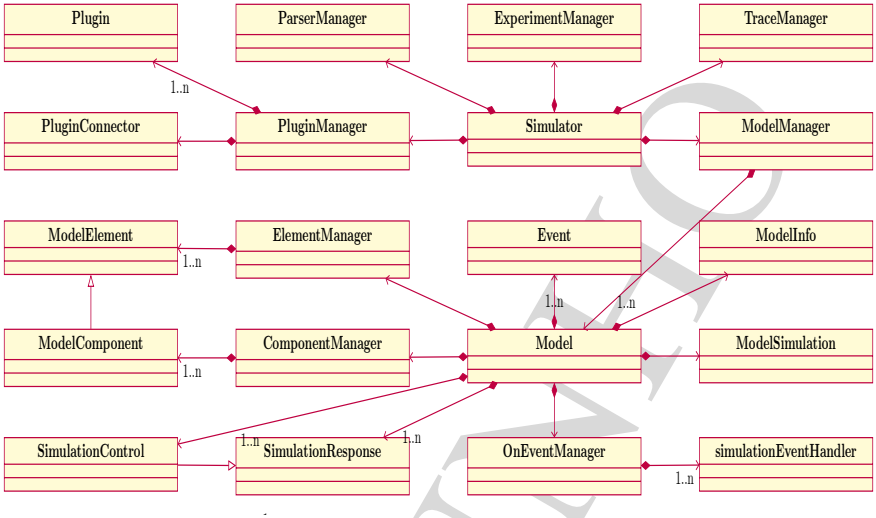

Figura 2.20: Principais classes do simulador

```
1  echo "Atualizando pacotes e instalando dependencias"
2  sudo apt-get update
3  sudo apt-get -y upgrade
4  sudo apt-get -y install build-essential git #gcc g++
5  sudo apt-get -y install qt5-default qtcreator libqt5designer5 #qt5-qmake
6  echo "Clonando repositorio do GenESyS"
7  git clone https://github.com/rlcancian/Genesys-Simulator.git
```

Listing 2.9: Comandos para instalar dependências necessárias para compilar o GenESyS

As principais pastas que estão no repositório do GenESyS são as seguintes:

- documentation (documentação doxygem gerada para a versão atual)
- models (arquivos textuais em linguagem de simulação do GenESyS com exemplos de modelos)
- projects (diferentes projetos para compilar parser, plugins, kernel, aplicações, ferraments, etc)
- source (código-fonte do GenESyS, seguindo a estrutura da figura 2.19).
  - applications
  - kernel
  - parser

  - plugins
- tests (pequenos testes unitários de funcionalidades esfecíficas)

Há diferentes formas de compilar o Genesys, dependendo se ele será usado (1) como uma biblioteca estática noutra aplicação qualquer, (2) como aplicação textual em terminal (console), ou (3) como uma aplicação gráfica (GUI). Além disso, pode-se compilar apenas o núcleo de simulação (kernel), apenas um plugin, apenas o parser, apenas uma ferramenta ou uma aplicação toda. Na versão dos estudantes as compilações que geram bibliotecas foram desabilitadas para facilitar a depuração, assim como a ligação dinâmica de plugins de simulação e do parser que interpreta expressões existentes nos modelos de simulação. A localização do arquivo (biblioteca ou executável) resultante depende do projeto usado na compilação.

Os comandos apresentados na listagem 2.10 podem ser usados compilar o GenESys e gerar o arquivo executável de uma aplicação textual (terminal ou console) que usa o simulador. Existem vários exemplos prontos de modelos e cuja aplicação associada simplesmente gera o modelo e o simula (ou seja, gera um *simulador específico* para aquele modelo). O usuário interessado pode selecionar um deles editando o arquivo de configuração em `source/applications/TraitsApp.h`.

```
1 echo "Compilando e executando aplicacao em terminal do GenESyS"
2 cd Genesys-Simulator
3 cd projects/GenesysTerminalApplication
4 make QMAKE=qmake
5 ./dist/Debug/GNU-Linux/genesysterminalapplication
```

Listing 2.10: Comandos para compilar e executar o GenESyS

Uma aplicação gráfica (GUI) sendo desenvolvida em QT pode ser compilada e executada a partir dos comandos da listagem 2.11. Essa interface gráfica é funcional e pode ser usada para criar modelos em linguagens de simulação do GenESyS ou criar modelos graficamente, com atualização simultânea do código em linguagem de simulação correspondente. A principal limitação no momento é a ausência de um Editor de Propriedades (*Property Editor*) que permita a edição dos parâmetros dos componentes do modelo. Interessados podem fazer contrbuições no código por *pull requests* no GitHub.

```
1 echo "Compilando e executando aplicacao grafica do GenESyS"
2 cd Genesys-Simulator
3 cd source/applications/gui/qt/GenesysQtGUI/build-GenesysQtGUI-Desktop-Debug/
4 make QMAKE=qmake
5 ./GenesysQtGUI
```

Listing 2.11: Comandos para compilar e executar o GenESyS

Ao invés de usar comandos pelo console, o autor sugere que se use a IDE *Apache Netbeans* 13.0 (ou mais recente) ou QtCreator 6.0.2 (ou mais recente) para trabalhar com o GenESyS em nível de desenvolvedor e como interessado no funcionamento de simuladores de sistemas. O código-fonte do GenESyS é organizado em pastas lógicas pelo NetBeans, o que facilita a compreensão de sua estrutura e organização.

Para instalar o NetBeans, execute o comando `sudo apt-get -y install libcanberra-gtk-module netbeans` ou faça o download da versão mais atual no website da *Apache*. Após instalado, abra o NetBeans e realize os seguintes passos para habilitar projetos em C++: (1) Acesse o Menu `Tools`,`Plugins`. Vá na aba `Settings` e ative a opção `Netbens 8.2 Plugin Portal`. (2) Na aba `Available Plugins` clique no botão `Check for Newest` e então instale o plugin `C/C++`. (3) Na aba `Installed` ative todos os plugins da categoria "C/C++"(CPPLiteKit e C/C++). Se faltar o *unpack200* para instalação dos plugins, ele pode ser encontrado no pacote `openjdk-13-jre-headless`.

Com projetos em C++ habilitados no Netbeans, você pode abrir o projeto do GenESyS. Para isso, vá no menu `File`, `Open Project` e selecione a pasta com o clone do repositório do GenESyS no github (`genesys`). Na janela de projetos do Netbeans aparecerá o projeto "Genesys-simulator". A estrutura de pastas que aparece nessa janela é lógica, e não física (não corresponde exatamente aos subdiretórios existentes no seu dispositivos de armazenamento). Você já pode começar a usar/programar/desenvolver esse simulador pelo NetBeans. A instalação do QtCreator é ainda mais fácil. Se você executou os comandos da listagem 2.9 então ele já está instalado. Ao executar o QtCreator você pode abrir o projeto da interface gráfica do GenESyS que já está disponível no código do GenESyS, na subpasta `source/applications/gui/qt/GenesysQtGUI`.

Primeiro Exemplo

O GenESyS pode ser usado como simulador de alto nível, como biblioteca de simulação ou como linguagem de simulação de baixo nível. Como na versão atual (2021.1) a interface gráfica (em Qt) ainda está sendo desenvolvida pelo autor, neste primeiro exemplo o GenESyS será usado como biblioteca de simulação para criar uma aplicação que corresponderá a um simulador de modelo específico bem simples. Além disso, como na versão atual a ligação dinâmica de plugins está desabilitada para fins didáticos de depuração do código, a inclusão de plugins é feita "artificialmente"(*fake*) e para facilitar esse processo (entre outros, como a inclusão de *traces*) o GenESyS já inclui uma classe-base para aplicações que usem o GenESyS como biblioteca de simulação, que é `BaseConsoleGenesysApplication`.

Iniciamos esse primeiro exemplo criando uma especialização dessa classe, que

chamaremos de `Example01`, como mostrado no trecho de código 2.12. A localização relativa da superclasse depende da pasta em a aplicação foi criada. Essa classe precisará apenas definir o método `main`, que conterá o código da aplicação. Em seguida editamos o arquivo de configuração `Traits.h` definindo essa classe como a aplicação a ser compilada, como mostra o trecho de código 2.13.

```
1 #include "../../BaseConsoleGenesysApplication.h"
2 class Example01 : public BaseConsoleGenesysApplication {
3   public:
4     Example01();
5     virtual int main(int argc, char** argv);
6 };
```

Listing 2.12: Arquivo Example01.h - Header da classe que será uma aplicação usando GenESyS

```
1 [...]
2 #include "userInterfaces/examples/book/Example01.h"
3 [...]
4 template <> struct Traits<GenesysApplication_if> {
5   static const Util::TraceLevel traceLevel = Util::TraceLevel::L6_arrival;
6   typedef Example01 Application;
7 };
```

Listing 2.13: Arquivo Trais.h - Configuração do GenESyS para compilar a aplicação Example01

O próximo passo é escrever o código da aplicação em si, que utilizará as classes do GenESyS para um programa que simule um modelo específico, como mostra o código 2.14 do arquivo `Example01.cpp`. Nesse código podemos ver a inclusão das bibliotecas utilizadas (linhas 2-6) e a implementação do método `main` (10-27). Na linha 11 uma instância (nomeada `genesys`) do simulador é criada. As linhas 12-13 invocam métodos da classe `BaseConsoleGenesysApplication` e servem para incluir artificalmente os plugins do simulador e vincular métodos para saída do simulador ao console. Na linha 14 um novo modelo é criado no simulador e uma referência para ele é retornada (`model`). A partir daí esse modelo de simulação é definido.

```
1   #include "Example01.h"
2   #include "../../../kernel/simulator/Simulator.h"
3   #include "../../../kernel/simulator/EntityType.h"
4   #include "../../../plugins/components/Create.h"
5   #include "../../../plugins/components/Delay.h"
6   #include "../../../plugins/components/Dispose.h"
7
8   Example01::Example01() { }
```

```
9
10  int Example01::main(int argc, char** argv) {
11     Simulator* genesys = new Simulator();
12     this->insertFakePluginsByHand(genesys);
13     this->setDefaultTraceHandlers(genesys->getTracer());
14     Model* model = genesys->getModels()->newModel();
15     EntityType* entityType1 = new EntityType(model, "Representative_Entity");
16     Create* create1 = new Create(model);
17     create1->setEntityType(entityType1);
18     create1->setTimeBetweenCreationsExpression("NORM(5,2)");
19     Delay* delay1 = new Delay(model);
20     delay1->setDelayExpression("unif(3,7)");
21     Dispose* dispose1 = new Dispose(model);
22     create1->getConnections()->insert(delay1);
23     delay1->getConnections()->insert(dispose1);
24     model->getSimulation()->setReplicationLength(30);
25     model->getSimulation()->start();
26     return 0;
27  }
```

Listing 2.14: Arquivo Example01.cpp - Código da aplicação Example01

Esse modelo é constituído por três componentes: um `Create`, um `Delay` e um `Dispose`, criados nas linhas 16, 19 e 21, respectivamente. O `Create` insere entidades no modelo e é parametrizado nas linhas 17-18, que definem o tipo de entidade criada e o tempo entre criações, que nesse caso é uma expressão que representa uma distribuição normal com média 5 e desvio 2 (unidades de tempo). O `Delay` representa um atraso de tempo, e é parametrizado na linha 20, em que o tempo de atraso segue uma distribuição uniforme com mínimo 3 e máximo 7 (u.t). Por fim, o `Dispose` retira entidades do modelo. As linhas 22-23 conectam esses três componentes, formando um "fluxo"ou "processo"que será percorrido por cada entidade. A linha 24 define a duração de uma simulação (replicação) em 30 (u.t) e a linha 25 inicia a simulação.

Ao ser executado, esse primeiro exemplo gera uma saída na tela que contém os trechos apresentados na listagem 2.15. Nela, destacamos o registro dos eventos que ocorreram durante a simulação/replicação (linhas 20-38), o relatório de estatísticas coletadas automaticamente pelo simulador (40-55) e o relatório que agrega estatísticas de cada uma das replicações (58-73).

```
1 STARTING GenESyS – GENeric and Expansible SYstem Simulator, version 210420 (russel)
2 |        LICENCE: Academic Mode. In academic mode this software has full functionality and executing
          training –size simulation models. This software may be duplicated and used for educational purposes
          only; any commercial application is a violation of the license agreement. Designed and developed by
          prof. Dr. Ing Rafael Luiz Cancian, 2018–2021
3 Component "Create_1" successfully inserted
4 Component "Delay_1" successfully inserted
5 Component "Dispose_1" successfully inserted
6 End of Model checking: Success
```

```
7  ------------------------------------------------------
8  GenESyS - GENeric and Expansible SYstem Simulator
9  LICENCE: Academic Mode. In academic mode this software has full functionality and executing training-size
       simulation models. This software may be duplicated and used for educational purposes only; any
       commercial application is a violation of the license agreement. Designed and developed by prof. Dr.
       Ing Rafael Luiz Cancian, 2018-2021
10 LIMITS: Based on your licence and activation code, your simulator is running under the following limits:
       50 components, 100 elements, 200 entities, 1 hosts, 1 threads.
11 Analyst Name:
12 Project Title:
13 Number of Replications: 1
14 Replication Length: 30.000000 second
15 Replication/Report Base TimeUnit: second
16 > Simulation controls: NumberOfReplications(ModelSimulation)=1.000000,
       ReplicationLength(ModelSimulation)=30.000000, WarmupPeriod(ModelSimulation)=360.000000,
       Create_1.EntitiesPerCreation(Create)=1.000000, Delay_1.Delay(Delay)=4.853544
17
18 Simulation of model "Model 1" is starting.
19 |
20 | Replication 1 of 1 is starting.
21 | | Event {time=0.000000, entity=1,comp="Create_1"}
22 | | Event {time=0.000000, entity=1,comp="Delay_1"}
23 | | Event {time=5.184605, entity=2,comp="Create_1"}
24 | | Event {time=5.184605, entity=2,comp="Delay_1"}
25 | | Event {time=5.633933, entity=1,comp="Dispose_1"}
26 | | Event {time=9.137748, entity=3,comp="Create_1"}
27 | | Event {time=9.137748, entity=3,comp="Delay_1"}
28 | | Event {time=10.746105, entity=2,comp="Dispose_1"}
29 | | Event {time=12.203979, entity=3,comp="Dispose_1"}
30 | | Event {time=17.148438, entity=4,comp="Create_1"}
31 | | Event {time=17.148438, entity=4,comp="Delay_1"}
32 | | Event {time=20.200767, entity=4,comp="Dispose_1"}
33 | | Event {time=24.486850, entity=5,comp="Create_1"}
34 | | Event {time=24.486850, entity=5,comp="Delay_1"}
35 | | Event {time=28.789898, entity=5,comp="Dispose_1"}
36 | Replication 1 of 1 has finished with last event at time 28.789898 second because replication length
       30.000000 second was achieved; Elapsed time 0.001222 seconds.
37 |
38 | Begin of Report for replication 1 of 1
39 | | | | name elems min max average
       variance stddev varCoef confInterv confLevel
40 | | Statistics for Create:
41 | | | Create_1:
42 | | | | Create_1.CountNumberIn ..... 5
43 | | Statistics for Delay:
44 | | | Delay_1:
45 | | | | Delay_1.WaitTime ..... 5 3.052329 5.633933 4.323408
       1.289380 1.135509 0.262642 0.995317 0.950000
46 | | Statistics for Dispose:
47 | | | Dispose_1:
48 | | | | Dispose_1.CountNumberIn ..... 5
49 | | Statistics for EntityType:
50 | | | EntityType1:
51 | | | | EntityType1.WaitTime ..... 5 3.052329 5.633933 4.323408
       1.289380 1.135509 0.262642 0.995317 0.950000
52 | | | | EntityType1.TotalTimeInSystem ..... 5 3.052329 5.633933 4.323408
       1.289380 1.135509 0.262642 0.995317 0.950000
53 | End of Report for replication 1 of 1
54 | ----------------------------------------
55 | Begin of Report for Simulation (based on 1 replications)
```

```
56 | | | | name                                   elems     min        max        average
      variance   stddev     varCoef     confInterv  confLevel
57 | | Statistics for Create:
58 | | | Create_1:
59 | | | | Create_1.CountNumberIn ................ 1  5.000000  5.000000  5.000000
      0.000000   0.000000   0.000000   0.000000   0.950000
60 | | Statistics for Delay:
61 | | | Delay_1:
62 | | | | Delay_1.WaitTime ...................... 1  4.323408  4.323408  4.323408
      0.000000   0.000000   0.000000   0.000000   0.950000
63 | | Statistics for Dispose:
64 | | | Dispose_1:
65 | | | | Dispose_1.CountNumberIn ............... 1  5.000000  5.000000  5.000000
      0.000000   0.000000   0.000000   0.000000   0.950000
66 | | Statistics for EntityType:
67 | | | EntityType1:
68 | | | | EntityType1.WaitTime .................. 1  4.323408  4.323408  4.323408
      0.000000   0.000000   0.000000   0.000000   0.950000
69 | | | | EntityType1.TotalTimeInSystem ......... 1  4.323408  4.323408  4.323408
      0.000000   0.000000   0.000000   0.000000   0.950000
70 | End of Report for Simulation
71 Simulation of model "Model 1" has finished. Elapsed time 0.001993 seconds.
```

Listing 2.15: Resultado da execução do Example01

A partir desse relatório, podemos ver, por exemplo, que ao longo da simulação foram criadas 5 entidades (linha 44) e que também 5 entidades saíram do sistema (linha 50), que o tempo de espera das estidades (linha 53) foi avaliado a partir de 5 entidades e os valores foram min=3.052329 (mínimo), max=5.633933 (máximo), average=4.323408 (média), variance=1.289380 (variância), stddev=1.135509 (desvio-padrão), varCoef=0.262642 (coeficiente de variação), confInterv=0.995317 (semi-intervalo de confiança), e confLevel=0.950000 (nível de confiança). Esse é o único tempo que as entidades ficam no sistema; por isso o tempo total das entidades no sistema (linha 54) possui os mesmos valores. O relatório de toda a simulação (58-73) é baseado numa única replicação, por isso os valores mínimo, máximo e média são iguais e os valores de dispensão (variância, desvio-padrão, etc) são todos zero.

O relatório de resultados do GenESyS é muito parecido com o relatório de saída do SIMAN (ver código 2.5), e como pode ser percebido, várias estatísticas são geradas automaticamente, sem que o projetista precise especificá-las no modelo. Outros exemplos prontos podem ser encontrados com o GenESyS.

2.4 Exercícios Propostos

Conceituação

Exercício 2.1 Quais são as diferenças entre uma linguagem de programação, uma linguagem de simulação e uma biblioteca de simulação?

Exercício 2.2 Cite casos em que o uso de linguagem de programação de propósito geral é mais indicado para criar modelos de simulação que o uso de linguagens de simulação. Cite casos em que o inverso é verdade.
Exercício 2.3 Quais é a diferença entre diagrama do modelo e declaração do modelo?
Exercício 2.4 O que é a modelagem baseada em atores? Cite exemplos de simuladores ou lingagens que utilizem essa forma de modelagem.

Aplicação
Exercício 2.5 Abra o exemplo "HiddenMarkovModelAnalysis" do Ptolemy II e mude a distrição de probabilidade que leva do estado x_1 ao x_2 e observe o resultado no histograma gerado.
Exercício 2.6 Altere o modelo de disseminação de doenças no Scicos para incluir uma nova equação diferencial e um novo parâmetro. Assuma que a taxa de infectados (x_2) continua aumentando à taxa de infecção e proporcional à fração de pessoas suscetíveis e infectadas, continua diminuindo proporcionalmente à taxa de recuperção e da fração de pessoas infetadas, mas que agora também diminui proporcionalmente a uma nova taxa de mortalidade γ e a fração de pessoas infectadas. A nova equação x_4 refere-se à fração de pessoas mortas em decorrência da doença, que aumenta proporcionalmente à taxa de mortalidade e da fração de pessoas infectadas. Assuma que $x_4(0) = 0$ e que $\gamma = 0.01$ e simule esse modelo por $t = 30$.
Exercício 2.7 Abra o exemplo "Banking Transactions" do Arena. Vá no menu "Rodar", então "SIMAN"e "Visualizar". Observe o código dessa linguagem de simulação (extensão .mod), que é gerado automaticamente pelo Arena através da tradução do modelo de alto nível (gráfico). Compare a versão gráfica do modelo com o modelo em linguagem de simulação SIMAN e identifique no modelo onde é especificado o tempo de atendimento dos caixas no banco.
Exercício 2.8 Abra o exemplo "Banking Transactions" do Arena e tente modificar o tempo de atendimento dos caixas do banco para produzir como saída da simulação uma quantidade média de cerca de 3 clientes na fila dos caixas.
Exercício 2.9 Abra o exemplo "Book_Cap02_Example01" do GenESyS, que foi apresentado na última seção, e modifique o tempo de espera para a expressão "Expo(2)"e anote os valores da média, desvio-padrão e coeficiente de variação do tempo total das entidades nesse modelo.

Investigação
Exercício 2.10 Faça uma pesquisa sobre outras linguagens de simulação existentes e em uso ainda hoje. Informe seus domínios de aplicação, aplicações principais e se há simuladores abertos que interpretam essas linguagens.
Exercício 2.11 Faça uma pesquisa sobre outros simuladores genéricos existentes e em uso ainda hoje, focando em simuladores gratuitos e de código aberto. Informe seus domínios de aplicação, aplicações principais, plataformas suportadas e outras

características de interesse.

Exercício 2.12 Faça uma pesquisa sobre os blocos para construção de modelos que estão disponíveis no Arena e verifique quais deles também existem no GenESyS e quais deles não existem no GenESyS.