Uma linguagem de *script* para o Mercury

Danilo Mendonça Oliveira
Orientador: Paulo Romero Martins Maciel

Universidade Federal de Pernambuco

3 de maio de 2014

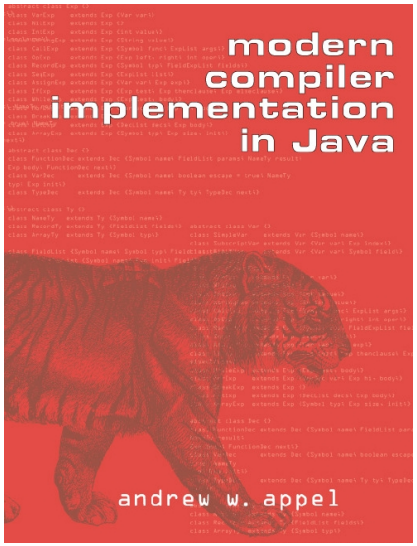
Roteiro

Introdução

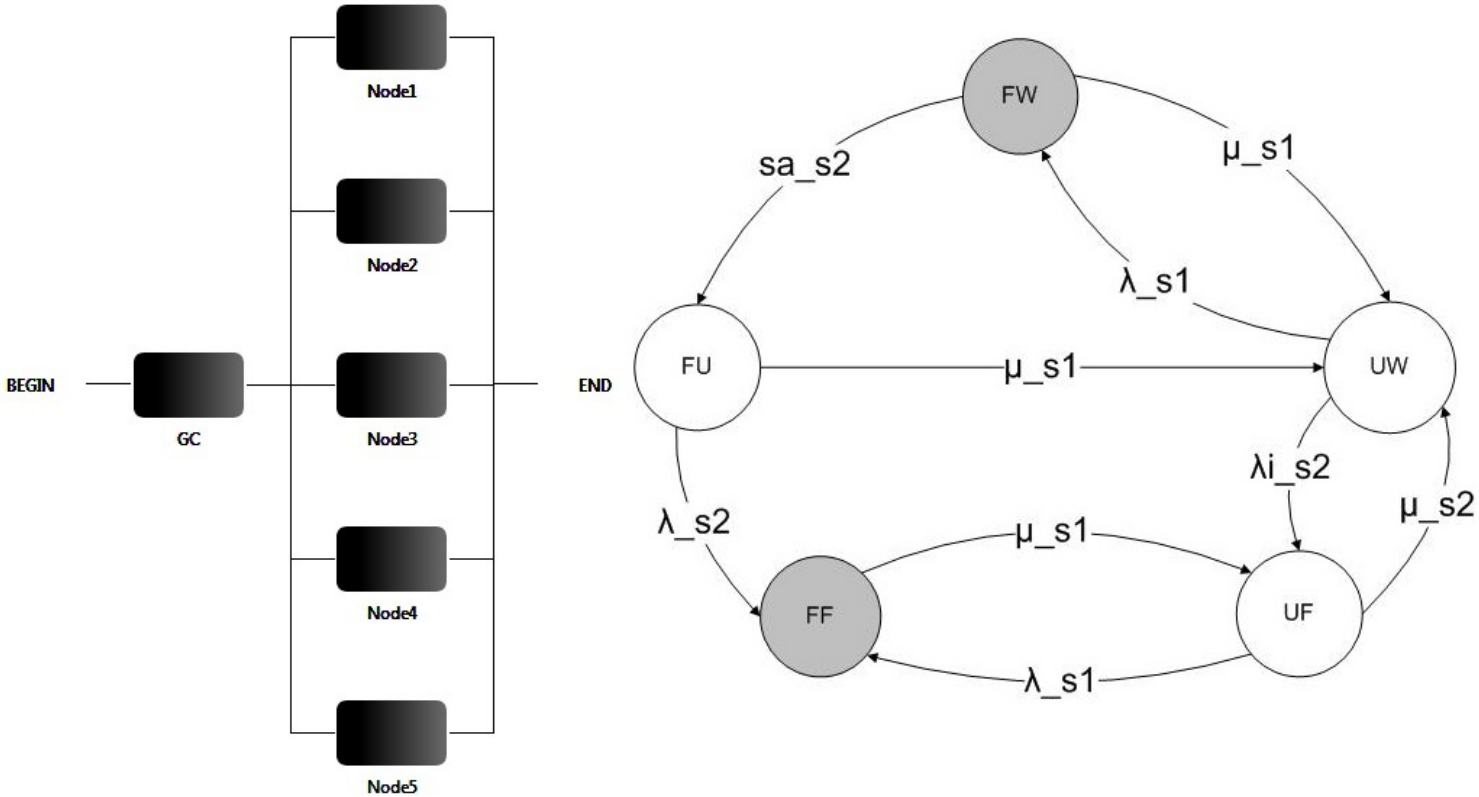
- Mercury CLI (*Command Line Interface*) - Interpretador e linguagem de *script* para a avaliação de modelos no Mercury
- Objetivos da Mercury CLI
  - Oferecer uma forma alternativa de usar o Mercury
  - Oferecer suporte à modelagem **hierárquica**
  - Oferecer suporte à avaliação **simbólica**

Materiais e métodos

- Análise Léxica: JFlex
- Análise Sintática: CUP
- Projeto de compilador baseado em:



- Estudo de caso extraído de: *Models for Dependability Analysis of Cloud. Computing Architectures for Eucalyptus. Platform. J. Dantas, R. Matos, J. Araujo and P. Maciel*



Modelo em cadeia de markov:

```
markov RedundantGC{
    state fu up;
    state fw;
    state ff;
    state uf up;
    state uw up;

    transition fw -> fu(rate = sa_s2);
    transition fu -> ff(rate = lambda_s2);
    transition ff -> uf(rate = mu_s1);
    transition uf -> uw(rate = mu_s2);
    transition uw -> fw(rate = lambda_s1);

    transition fw -> uw(rate = mu_s1);
    transition uw -> uf(rate = lambdai_s2);
    transition uf -> ff(rate = lambda_s1);

    transition fw -> ff(rate= lambdai_s2);
    transition fu -> uw(rate = mu_s1);

    metric aval = availability;
}
```

Modelos em RBD:

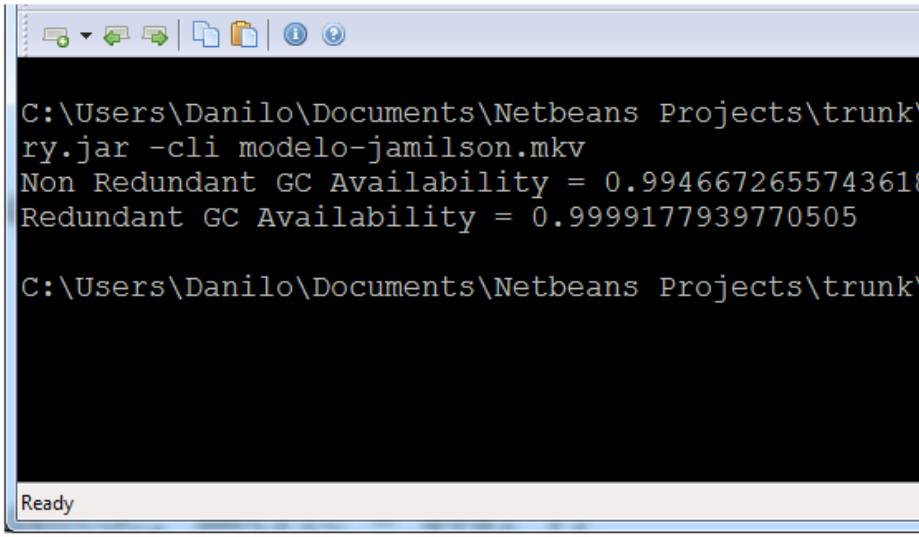
```
RBD NonRedundantGC{
    block hw(MTTF = mttfhw, MTTR = mttrhw);
    block so(MTTF = mttfso, MTTR = mttrso);
    block clc(MTTF = mttfclc, MTTR = mttrclc);
    block cc(MTTF = mttfcc, MTTR = mttrcc);
    block sc(MTTF = mttfsc, MTTR = mttrsc);
    block walrus(MTTF = mttfwalrus, MTTR = mttrwalrus);

    series s1(hw, so, clc, cc, sc, walrus);

    top s1;

    metric aval = availability;
}

RBD Node{
    block hw(MTTF = mttfhw, MTTR = mttrhw);
    block so(MTTF = mttfso, MTTR = mttrso);
    block kvm(MTTF = mttfkvm, MTTR = mttrkvm);
    block nc(MTTF = mttfnc, MTTR = mttrnc);

    series b1(hw, so, kvm, nc);
```

Modelo hierárquico em RBD:

```
RBD RedundantCloud{
    hierarchy gc(availability=solve(model = RedundantGC, metric = aval) );

    hierarchy node1(availability=solve(model = Node, metric = aval));
    hierarchy node2(availability=solve(model = Node, metric = aval));
    hierarchy node3(availability=solve(model = Node, metric = aval));
    hierarchy node4(availability=solve(model = Node, metric = aval));
    hierarchy node5(availability=solve(model = Node, metric = aval));

    parallel b1(node1, node2, node3, node4, node5);

    series b2(gc, b1);

    top b2;

    metric aval = availability;
}
```

Avaliando os modelos:

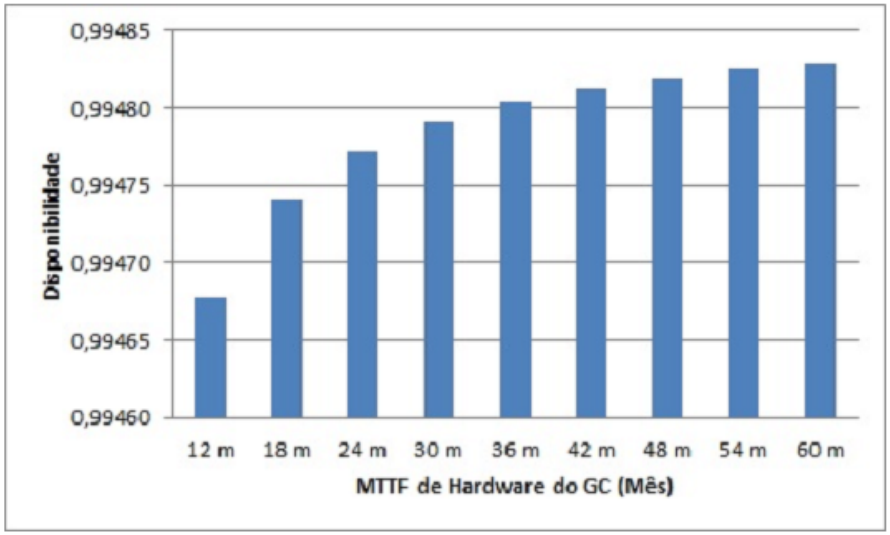
```
main{
    lambda_s1 =  1/180.7212397;
    mu_s1 = 1/0.966902178;
    mu_s2 = 1/0.966902178;
    lambdai_s2 = 1/216.8654876049552;
    lambda_s2 = 1/180.7212397;
    sa_s2 = 1/0.005555555;

    (...)

    a = solve( model = NonRedundantGC, metric = aval );
    println( "Non Redundant GC Availability = " .. a );

    a2 = solve( model = RedundantGC, metric = aval );
    println( "Redundant GC Availability = " .. a2 );

}
```



Usando o modo interativo do interpretador:

```
C:\Users\Danilo\Documents\Netbeans Projects\trunk\dist>java -jar Mercury.jar -cl
i -i modelo-jamilson.mkv
Non Redundant GC Availability = 0.9946672655743618
Redundant GC Availability = 0.9999177939770505
Non redundant cloud = 0.9946672655743386
>mttfhw = mttfhw * 2;
>a = solve( model = NonRedundantCloud, metric = aval );
>println(a);
0.9947618786501141
>
```

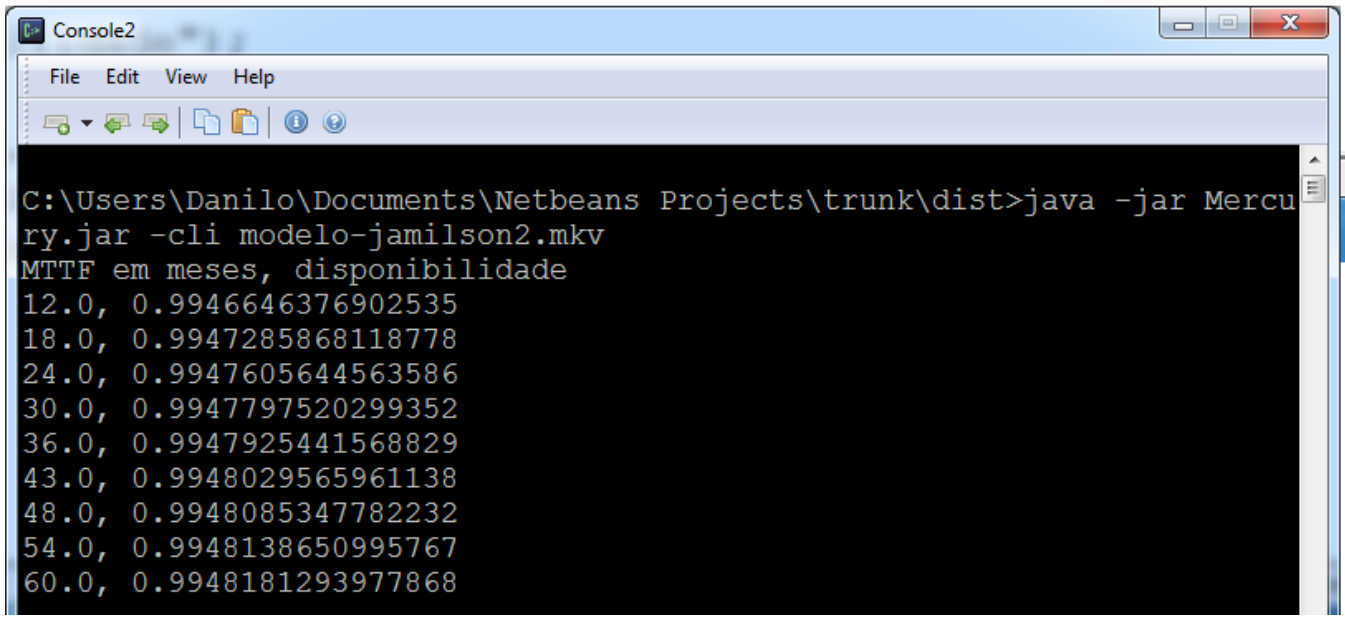
Realizando experimentos:



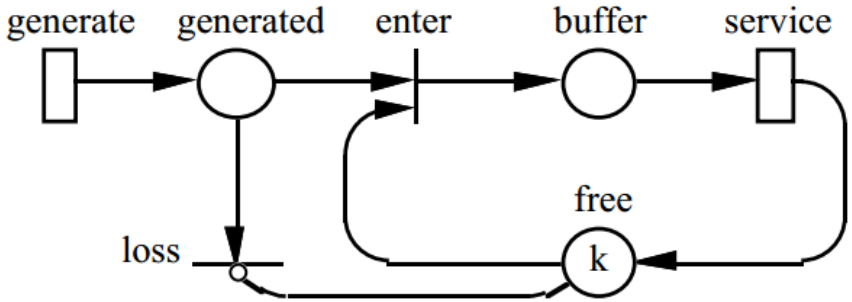
(a) Arquitetura não Redundante

Realizando experimentos:

```
println("MTTF em meses, disponibilidade");

for mttfmeses in [12, 18, 24, 30, 36, 43, 48, 54, 60]{
    mttfhw = 30 * 24 * mttfmeses;
    a = solve( model = NonRedundantCloud, metric = aval );
    println(mttfmeses .. ", " .. a );
}
```



- Estudo de caso extraído de: *German, R. (1996), A concept for the modular description of stochastic petri nets , in 'Proc. 3rd Int. Workshop on Performability Modeling of Computer and Communication Systems'*



Modelando a rede de Petri via script:

```
SPN Foo{
    place gerados;
    place buffer;
    place livres(tokens = 10);

    timedTransition gerar(
        delay = 1,
        outputs = [gerados]
    );

    timedTransition servir(
        delay = servir,
        inputs = [buffer],
        outputs = [livres],
        serverType = "ExclusiveServer"
    );
```

(continuação)

```
    immediateTransition descarta(
        inputs = [gerados],
        inhibitors = [livres]
    );

    immediateTransition enfileira(
        inputs = [gerados, livres],
        outputs = [buffer]
    );

    metric m1 = stationaryProbability( expression = "P{#buffer>0}" );
}
```

Considerações finais

- O CLI do Mercury junto com a linguagem de script desenvolvida fornecem um mecanismo bastante flexível para a análise de modelos, aumentando a produtividade do usuário
- Não existe pretensão em tornar esta linguagem de script uma linguagem de programação mais completa
- Uma interface gráfica que gerasse os scripts poderia dar acesso ao melhor dos dois mundos

Trabalhos futuros...

- ~~Interface de "console interativo"~~
- Mecanismo de "herança" para criar novos modelos baseados em modelos existentes
- Comando "include" para permitir scripts menores e mais organizados
- Arquivo de configuração para os parâmetros da avaliação de CTMCs(erro, algoritmo, etc.)
- Avaliação de RdPs por simulação
- Sintaxe especial para distribuições "Phase-Type"
- ~~Permitir que as variáveis armazene diferentes tipos de dados, inclusive dados estruturados~~
- ~~Comando condicional e expressões booleanas~~
- ~~Funções definidas pelo usuário~~