4
Linguagens de Modelagem de IHC

A ferramenta proposta no atual trabalho visa englobar algumas linguagens visuais comuns na área de Interação Humano-Computador. São elas: Concur Task Trees (CTT), MoLIC e Statecharts.

4.1
Concur Task Trees (CTT)

O ConcurTaskTrees (CTT) é uma linguagem utilizada para representar a modelagem de tarefas, que se concentra no projeto e especificações de aplicações seguindo-se um roteiro que combina estruturas hierárquica de tarefas concorrentes com um conjunto de operadores temporais. Esta notação representa atividades interativas, através da decomposição de tarefas numa estrutura de árvore invertida [Paternò *et al.*, 1999].

A Figura 20 mostra um modelo CTT. O foco no CTT se dá principalmente nas relações de precedência entre as tarefas.

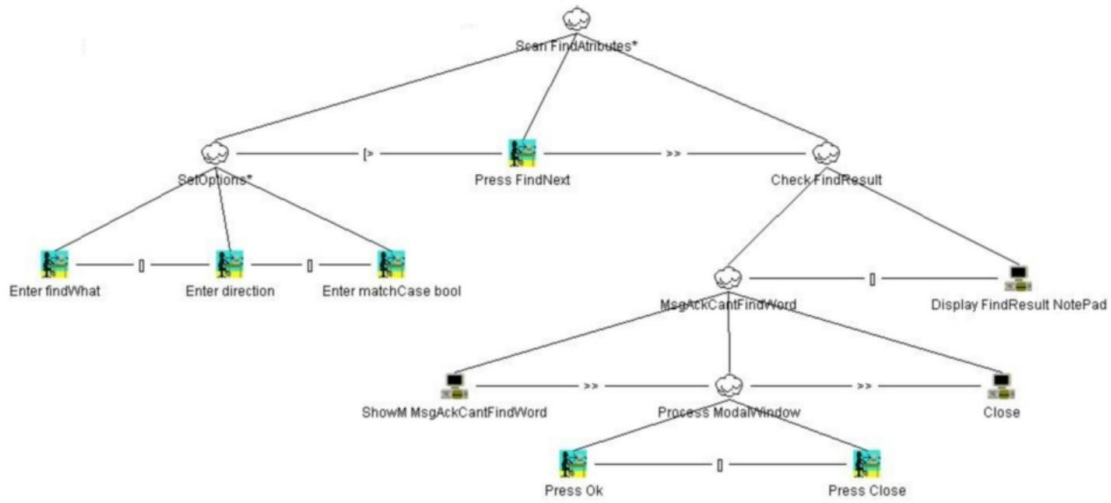

Figura 20 - Diagrama do CTT.

Na Figura 20, é apresentado um modelo de tarefas utilizando o CTT. O modelo de tarefas é uma decomposição hierárquica de sub-tarefas que devem ser seguidas para se atingir um objetivo. O objetivo está na raíz da árvore. As folhas representam a atual interação entre o usuário e o dispositivo. Existem quatro tipos diferentes de tarefas no modelo:

- Tarefas interativas  representam entradas do usuários à aplicação;
- Tarefas da aplicação  representam a saída da aplicação para o usuário;
- Tarefas dos usuários  representam os pontos de decisão da parte do usuário;
- Todas as tarefas anteriores devem aparecer como folhas no modelo, tarefas abstratas  são usadas para estruturar o modelo e aparecem como nós internos na árvore.

O problema de se identificar tarefas do usuário e da aplicação é simplificada através da classificação.

A sub-árvore apresentada na figura pertence ao modelo de tarefas utilizado para encontrar texto em um editor de texto. A semântica do modelo é definida pelos possíveis percursos na árvore. O percurso na árvore é realizado da esquerda para a direita, através de uma busca em profundidade e é governado por operadores aplicados a tarefas ou a pares de tarefas que encontram-se no mesmo nível. Agora nós introduzimos o operador mais relevante para descrever o relacionamento induzido pela árvore na figura. Primeiramente, nota-se que o topo da tarefa (*ScanFindAttribute*) é interativo (operador de tarefa interativa: *), o que significa que ela pode ser executada repetidamente. Para que ela seja executada, *SetOptions* deve ser executada (sendo ela mesma uma tarefa interativa). Neste ponto, o usuário pode escolher entre as três alternativas possíveis na árvore (operador de escolha: []): *Enter findWhat, Enter direction,* ou *Enter matchCase bool.* A qualquer momento na execução de *SetOptions* o usuário pode escolher executar a tarefa *Press FindNext* (operador de desabilitação: [>), assim que terminar a tarefa anterior. Após *Press FindNext*, a tarefa *Check FindResult* se tornará possível (operador

de permissão: >>). O diálogo continua descendo pela sub-árvore *Check FindResult.*

A descrição acima não inclui todos os operadores CTT. Operadores adicionais relevantes no presente contexto são os operadores de tarefas opcionais (expressão “[T]” significa que a tarefa T é opcional), e o operador que suspende a retomada (expressão “T1 |> T2” significa que o T2 interrompe T1, mas T1 continuará de onde ele foi interrompido após T2 ser finalizado).

Segundo Nunes [Nunes,2001], o CTT especificamente é adotado pela maioria das abordagens baseadas em modelos para IHC atualmente. Entretanto, existem algumas lacunas na notação e em sua interpretação que fazem com que um trabalho com a MoLIC seja significativo no que diz respeito a melhores contribuições, sendo o tratamento de erros uma delas. Isto porque, não fica claro na modelagem de tarefas quais delas servem para a recuperação de erros. Outra lacuna seria a incapacidade de se visualizar todos os caminhos necessários para se atingir todos os objetivos do usuário. Desta forma, acredita-se que a complementação da modelagem de tarefas através de um modelo de diálogo como a MoLIC possa contribuir de forma positiva para um grau de expressão maior na interação que será experimentada pelos usuários.

4.1.1
Representação da CTT na linguagem de regras

A primeira parte do metamodelo define os seus elementos nós e arestas. Os nós são representados por 4 elementos em formato PNG ou GIF: *interaction task, aplication task, user task* e *abstract task* (Figura 21).

```
<xml>
    <metamodel name="ctt">
        <elements>
            <nodes>
                <node id="application" width="80" height="80" stylenode="image"
                      imagename="aplicacao.gif"/>
                <node id="user" width="80" height="80" stylenode="image"
                      imagename="user.png" />
                <node id="interaction" width="80" height="80" stylenode="image"
                      imagename="interaction.png" />
                <node id="abstract" width="80" height="80" stylenode="image"
                      imagename="abstract.png" />
            </nodes>
```

Figura 21 - Nós da linguagem CTT.

As arestas são representadas por 9 elementos: *parent*, *choice*, *order independence*, *interleaving*, *synchronization*, *disabling*, *suspend resume*, *sequential enabling* e *sequential enabling info*. As arestas são criadas apenas no metamodelo. Elas não necessitam de imagens (Figura 22).

```
<edges>
    <edge id="parent" value="" width="120" height="120" fontcolor="#000000" >
        <startarrow  type="none" />
        <body  type="straight" color="#0000FF" strokewidth="2" dashed="0" />
        <endarrow  type="none" />
    </edge>
    <edge id="choice" value="[]" width="120" height="120" fontcolor="#000000" >
        <startarrow  type="none" />
        <body  type="straight" color="#000000" strokewidth="2" dashed="0" />
        <endarrow  type="none" />
    </edge>
    <edge id="order-independence" value="|=|" width="120" height="120"
          fontcolor="#000000" >
        <startarrow  type="none" />
        <body  type="straight" color="#000000" strokewidth="2" dashed="0" />
        <endarrow  type="none" />
    </edge>
    <edge id="interleaving" value="|||" width="120" height="120"
          fontcolor="#000000" >
        <startarrow  type="none" />
        <body  type="straight" color="#000000" strokewidth="2" dashed="0" />
        <endarrow  type="none" />
    </edge>
```

```
        <edge id="synchronization" value="|[]|" width="120" height="120"
              fontcolor="#000000" >
            <startarrow  type="none" />
            <body  type="straight" color="#000000" strokewidth="2" dashed="0" />
            <endarrow  type="none" />
        </edge>
        <edge id="disabling" value="[>" width="120" height="120"
              fontcolor="#000000" >
            <startarrow  type="none" />
            <body  type="straight" color="#000000" strokewidth="2" dashed="0" />
            <endarrow  type="none" />
        </edge>
        <edge id="suspend-resume" value="|>" width="120" height="120"
              fontcolor="#000000" >
            <startarrow  type="none" />
            <body  type="straight" color="#000000" strokewidth="2" dashed="0" />
            <endarrow  type="none" />
        </edge>
        <edge id="sequential-enabling" value=">>" width="120" height="120"
              fontcolor="#000000" >
            <startarrow  type="none" />
            <body  type="straight" color="#000000" strokewidth="2" dashed="0" />
            <endarrow  type="none" />
        </edge>
        <edge id="sequential-enabling-info" value="[]>>" width="120" height="120"
              fontcolor="#000000" >
            <startarrow  type="none" />
            <body  type="straight" color="#000000" strokewidth="2" dashed="0" />
            <endarrow  type="none" />
        </edge>
    </edges>
</elements>
```

Figura 22 - Arestas da linguagem CTT.

Na parte das regras foram utilizadas as seguintes regras para definir o modelo CTT (Figura 23):

- Utilizando a aresta *parent,* o elemento *application* só se conecta ao elemento *application*.
- Utilizando a aresta *parent*, o elemento *user* só se conecta ao elemento user.
- Utilizando a aresta *parent*, o elemento *interaction* só se conecta ao elemento interaction.
- Utilizando a aresta *parent*, o elemento *abstract* se conecta a qualquer outro elemento.

```
        <rules type="graph">
            <rule element="parent" type="edge">
                <option negative="false">
                    <source>
                        <element id="application"  />
                    </source>
                    <target>
                        <element id="application" />
                    </target>
                </option>
                <option negative="false">
                    <source>
                        <element id="user"  />
                    </source>
                    <target>
                        <element id="user" />
                    </target>
                </option>
                <option negative="false">
                    <source>
                        <element id="interaction"  />
                    </source>
                    <target>
                        <element id="interaction" />
                    </target>
                </option>
                <option negative="false">
                    <source>
                        <element id="abstract"  />
                    </source>
                    <target>
                        <element id="user" />
                        <element id="application" />
                        <element id="interaction" />
                        <element id="abstract" />
                    </target>
                </option>
            </rule>
        </rules>
    </metamodel>
</xml>
```

Figura 23 - Regras relacionadas a aresta Parent.

4.2 MoLIC

A linguagem MoLIC (Modeling Language for Interaction as Conversation) é um dos metamodelos a serem analisados para a definição do XML Schema em que os metamodelos serão representados. Ela é uma linguagem de modelagem utilizada para representar a interação usuário–sistema seguindo a metáfora de uma conversa entre usuário e designer (sistema), com base na teoria da Engenharia Semiótica [Souza, 2005]. A MoLIC foi criada em 2003 [Paula & Barbosa, 2003] para apoiar a reflexão do designer ao elaborar as possíveis conversas que os usuários poderão travar através da interface do sistema.

4.2.1
Cena

Uma cena representa uma conversa entre o usuário e o preposto do de-*signer* sobre um certo assunto ou tópico. O tópico da cena está associado a uma tarefa do usuário. Ele indica as interações que o usuário pode realizar naquele momento. Uma cena pode conter diálogos que são compostos por falas do usuário e do designer.

Graficamente, uma cena possui duas representações: a cena expandida é representada por um retângulo com bordas arredondadas e uma linha horizontal interna separando o tópico dos diálogos (Figura 24), enquanto a cena mínima é representada por um retângulo com bordas arredondadas contendo apenas o seu tópico, sem diálogos (Figura 25).

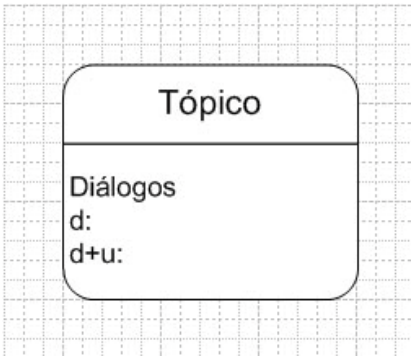

Figura 24 - Cena com diálogo.

Figura 25 - Cena sem diálogo.

4.2.2
Acesso Ubíquo

Ele representa o início de uma conversa em direção a um objetivo, e cujas falas de transição podem ser emitidas em qualquer momento durante a interação. O acesso ubíquo é representado por uma cena anônima de fundo cinza [Barbosa & Silva, 2010] (Figura 26).

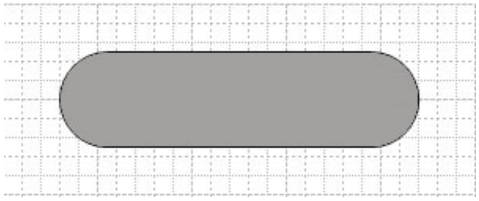

Figura 26 - Acesso Ubíquo

4.2.3
Ponto de Entrada

Também conhecido como ponto de abertura, indica o início da interação do usuário com o sistema. Ele é representado por um círculo preechido na cor preta (Figura 27).

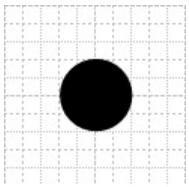

Figura 27 - Ponto de entrada.

4.2.4
Ponto de Saída

Também conhecido como ponto de encerramento, indica o final da interação do usuário com o sistema. Ele é representado por um círculo na cor preta circunscrito a um círculo branco (Figura 28).

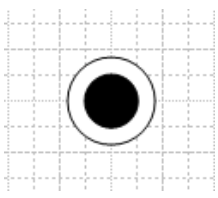

Figura 28 - Ponto de saída.

4.2.5
Processo do Sistema

Representado graficamente por uma caixa preta, o processo do sistema indica o turno do preposto do designer, ou seja, um processamento que está ocorrendo e que o usuário "não vê dentro" (Figura 29). Busca enfatizar a necessidade de comunicar, através de falas de transição do preposto do designer, o que ocorreu como resultado de uma solicitação do usuário.

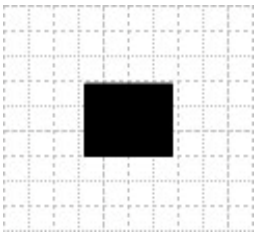

Figura 29 - Processo do sistema.

4.2.6
Fala de Troca de Turno ou Transição

A fala de troca de turno (ou transição) é utilizada para representar a troca de turno entre usuário e sistema e é representada por uma seta com linha sólida, quando não acontece nenhum erro na conversa do usuário com o preposto do designer. Quando emitida pelo usuário, ela deve partir de uma cena, e pode ter como destino uma outra cena ou um processamento do sistema (Figura 30, u:utterance). Quando emitida pelo sistema, ela deve partir de um processamento do sistema, e pode ter como destino uma cena ou um outro processamento do sistema (Figura 30, d:utterance).

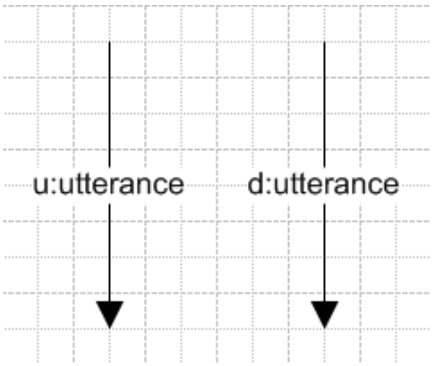

Figura 30 - Falas de troca de turno do usuário e do preposto designer.

4.2.7
Fala de recuperação de ruptura (*breakdown*)

Uma fala de recuperação de ruptura representa um momento na interação em que o usuário (Figura 31, u:utterance) ou o designer (Figura 31, d:utterance) identifica uma ruptura na conversa. Um exemplo de ruptura ocorre quando os valores informados pelo usuário em uma cena não são válidos e o designer lhe oferece uma chance de corrigi-los, informando o problema que ocorreu, ou quando o próprio usuário muda de ideia, possivelmente porque percebeu que a conversa não estava tomando o rumo esperado.

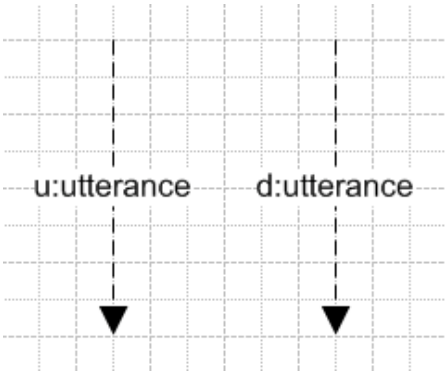

Figura 31 - Falas de recuperação de ruptura do usuário e do designer.

4.2.8
Representação da MoLIC na linguagem de regras

A primeira parte do metamodelo define os seus elementos nós e arestas. Os nós são representados por 5 elementos em formato SHAPE: ponto de entrada, acesso ubíquo, ponto de saída, cena e sistema (Figura 32).

```
<xml>
    <metamodel name="molic">
        <elements>
            <nodes>
                <node id="pontodeentrada" width="40" height="40" stylenode="shape"/>
                <node id="acessoubiquo" width="120" height="60" stylenode="shape"/>
                <node id="pontodesaida" width="40" height="40" stylenode="shape"/>
                <node id="cena" width="160" height="80" stylenode="shape" />
                <node id="sistema" width="40" height="40" stylenode="shape"/>
            </nodes>
```

Figura 32 - Nós da linguagem MoLIC.

As arestas são representadas por 3 elementos: fala de recuperação de ruptura, fala de troca de turno ou transição do designer e do usuário. As arestas são criadas apenas no metamodelo. Elas não necessitam de imagens (Figura 33).

```
    <edges>
        <edge id="U-Transicao" value="U:" width="120" height="120" imageicon="">
            <startarrow  type="none" />
            <body  type="horizontal" color="#0000FF" strokewidth="2" dashed="false" />
            <endarrow  type="classic" />
        </edge>
        <edge id="D-Transicao" value="D:" width="120" height="120" imageicon="">
            <startarrow  type="none" />
            <body  type="horizontal" color="#0000FF" strokewidth="2" dashed="false" />
            <endarrow  type="classic" />
        </edge>
        <edge id="D-Recuperacao" value="D:" width="120" height="120" imageicon="" >
            <startarrow  type="none" />
            <body  type="horizontal" color="#FF0000" strokewidth="2" dashed="true" />
            <endarrow  type="classic" />
        </edge>
    </edges>
</elements>
```

Figura 33 - Arestas da linguagem MoLIC.

Na parte das regras foram utilizadas as seguintes regras para definir o modelo MoLIC:

- O nó **Ponto de entrada** não possui arestas de entrada (Figura 34).
- O nó **Ponto de entrada** não pode se conectar a outro nó **Ponto de entrada** (Figura 34).

```
<rules type="graph" warningicon="true">
    <rule element="pontodeentrada" type="node" >
        <option negative="true">
            <source>
                <element id="pontodeentrada"  />
                <element id="pontodesaida" />
                <element id="acessoubiquo" />
                <element id="cena" />
                <element id="sistema" />
            </source>
            <target>
                <element id="pontodeentrada"  />
            </target>
        </option>
    </rule>
```

Figura 34 - Regra relacionada ao Ponto de entrada.

- O nó **Ponto de saída** não possui arestas de saída (Figura 35).
- O nó **Ponto de saída** não pode se conectar a outro nó **Ponto de saída** (Figura 35).

```
<rule element="pontodesaida" type="node" >
    <option negative="true">
        <source>
            <element id="pontodesaida" />
            <element id="pontodeentrada" />
        </source>
        <target>
            <element id="pontodesaida" />
            <element id="pontodeentrada" />
            <element id="acessoubiquo" />
            <element id="cena" />
            <element id="sistema" />
        </target>
    </option>
</rule>
```

Figura 35 - Regras relacionada ao Ponto de saída.

- A aresta **Fala de recuperação de ruptura** deve sair do nó **Processo do sistema** para o nó **Cena** (Figura 36).

```
<rule element="D-Recuperacao" type="edge" >
    <option negative="false">
        <source>
            <element id="sistema" />
        </source>
        <target>
            <element id="cena" />
        </target>
    </option>
</rule>
```

Figura 36 - Regra relacionada a Fala de recuperação de ruptura.

- A aresta **Fala de troca de turno ou transição do designer** deve sair do nó **Processo de sistema** para os nós **Cena**, **Ponto de saída** ou **Acesso ubíquo** (Figura 37).

```
<rule element="D-Transicao" type="edge" >
    <option negative="false">
        <source>
            <element id="sistema" />
        </source>
        <target>
            <element id="cena" />
            <element id="pontodesaida" />
            <element id="acessoubiquo" />
        </target>
    </option>
</rule>
```

Figura 37 - Regra relacionada a Fala de troca de turno ou transição do designer.

- A aresta **Fala de troca de turno ou transição do usuário** deve sair sempre dos nós **Cena**, **Ponto de entrada** e **Acesso ubíquo** para os nós **Cena**, **Ponto de saída**, **Processo do sistema** e **Acesso Ubíquo** (Figura 38).

```
            <rule element="U-Transicao" type="edge" >
                <option negative="false">
                    <source>
                        <element id="cena" />
                        <element id="pontodeentrada" />
                        <element id="acessoubiquo" />
                    </source>
                    <target>
                        <element id="cena" />
                        <element id="pontodesaida" />
                        <element id="sistema" />
                        <element id="acessoubiquo" />
                    </target>
                </option>
            </rule>

        </rules>
    </metamodel>
</xml>
```

Figura 38 - Regra relacionada a Fala de troca de turno ou transição do usuário.

4.3
Statecharts

Statecharts podem ser considerados diagramas de transição de estado estendidos, onde são mostrados os eventos e estados interessantes de um objeto e seu comportamento em reação a um evento [Harel, 1987]. São usados para modelar os aspectos dinâmicos do sistema, representando seus estados, classes ou casos de uso, sendo representado através de um fluxo de controle de um estado para o outro.

Os estados representam o contexto onde comportamentos ocorrem, formando o conjunto dos elementos do sistema. As transições ocorrem quando há mudança para um dado contexto e qual contexto é esse, sendo habilitada por um evento e uma condição.

Harel [Harel, 1987] formalizou o Statecharts como uma solução para os problemas combinatórios com diagramas de transição, consolidando o conceito de meta-state (meta estado). Meta estados agrupam conjuntos de estados com transições em comum que são herdados por todos os estados que são inclusos em meta estados.

Um estado de atividade pode fazer referência a outro diagrama de atividades que mostre a estrutura interna do estado de atividades. Se for especificado de outra forma, você poderá ter gráficos de atividades aninhados. É possível mostrar o subgráfico dentro do estado de atividade ou permitir que o estado de atividade faça referência a outro diagrama.

Existem dois tipos de meta-states. O tipo XOR possui transições sequenciais (Figura 39), onde há apenas um estado ativo por vez no meta-state. O tipo AND possui transições paralelas (Figura 40), onde há mais de um estado ativo dentro do meta-state. Há ainda o History state (H e H*) (Figura 41), que ocorrem apenas dentro de XOR meta-states. Este tipo ocorre sempre que uma transição transfere o controle de um meta-state, lembrando qual estado estava ativo antes dela acontecer. Ou seja, assim que uma transição devolver o controle para o meta-state, ele volta a partir da configuração lembrada. Um exemplo disso é a opção de “Ajuda” de um sistema.

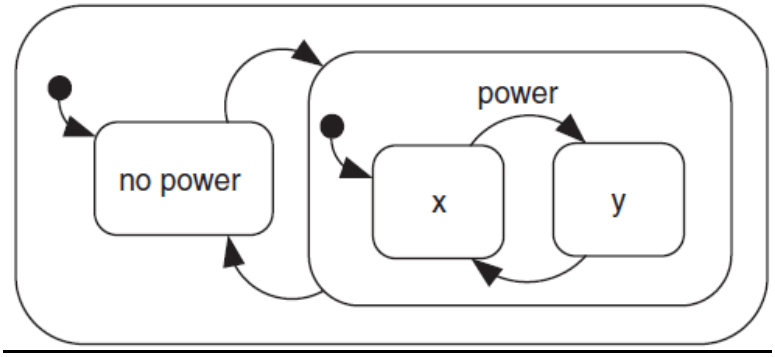

Figura 39 - Estados simples com XOR meta-state.

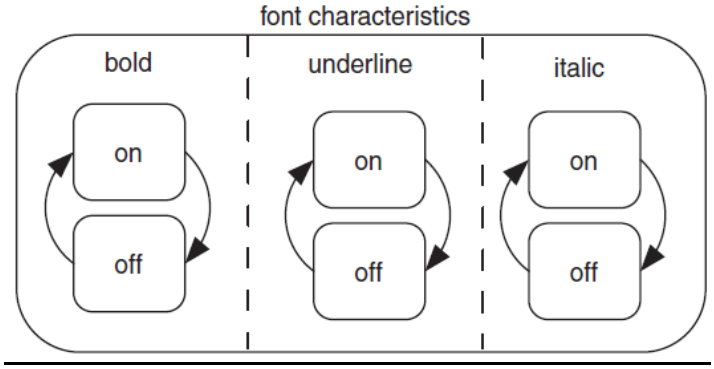

Figura 40 - Estado concorrente com AND meta-state.

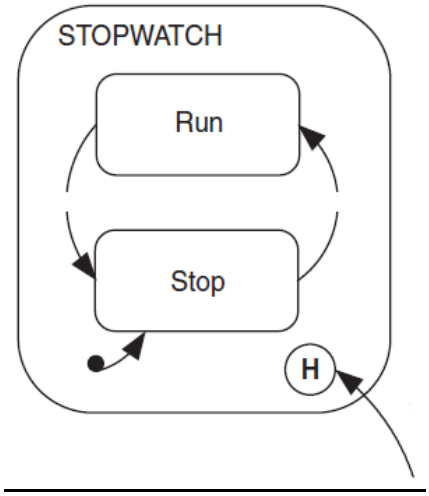

Figura 41 - History State.

4.3.1
Representação de Statecharts na linguagem de regras

A primeira parte do metamodelo define os seus elementos nós e arestas. Os nós são representados por 5 elementos em formato SHAPE: estado padrão, história, super-estado, sub-estado, estado básico (Figura 42).

```
<xml>
    <metamodel name="Statecharts">
        <elements>
            <nodes>
                <node id="estado-padrao" width="40" height="40" stylenode="shape"/>
                <node id="historia" width="40" height="40" stylenode="shape"/>
                <node id="superestado" width="120" height="80" stylenode="shape" />
                <node id="subestado" width="80" height="80" stylenode="shape" />
                <node id="estado-basico" width="80" height="80" stylenode="shape" />
            </nodes>
```

Figura 42 - Nós da linguagem Statecharts.

As arestas são representadas por um único elemento de nome “aresta”, pois na linguagem Statecharts só necessita de uma aresta dirigida. Ela é criada diretamente no metamodelo e não necessita de uma imagem (Figura 43).

```
    <edges>
        <edge id="aresta" value="" width="120" height="120" imageicon="">
            <startarrow  type="none" />
            <body  type="horizontal" color="#000000" strokewidth="2" dashed="false" />
            <endarrow  type="classic" />
        </edge>
    </edges>
</elements>
```

Figura 43 - Aresta da linguagem Statecharts.

Na parte das regras foram utilizadas as seguintes regras para definir o modelo Statecharts:

- O nó super-estado não pode conter o nó estado básico (Figura 44).

```
<rules type="graph" warningicon="true">
    <rule element="super-estado" type="node" >
        <option negative="true">
            <children>
                <element id="estado-basico"/>
            </children>
        <option/>
    </rule>
```

Figura 44 - Regra relacionada ao nó super-estado.

- O nó estado padrão só pode apontar para os nós: super-estado, sub-estado e estado básico (Figura 45).

```
<rule element="estado-padrao" type="node" >
    <option negative="false">
        <source>
        </source>
        <target>
            <element id="sub-estado"/>
            <element id="super-estado"/>
            <element id="estado-basico"/>
        </target>
    </option>
</rule>
```

Figura 45 - Regra relacionada ao nó estado padrão.

- O nó história só pode ser apontado pelos nós: sub-estado e estado básico (Figura 46).

```
            <rule element="historia" type="node" >
                <option negative="false">
                    <source>
                        <element id="sub-estado"/>
                        <element id="estado-basico"/>
                    </source>
                    <target>
                    </target>
                </option>
            </rule>
        </rules>
    </metamodel>
</xml>
```

Figura 46 - Regras relacionada ao nó história.