Framework para Simulação e Verificação de Aplicações Adaptativas

F. M. B. Reis and A. R. Camolesi

***Abstract*— Este trabalho tem por objetivo apresentar o conceito e a implementação de um framework para o desenvolvimento de aplicações adaptativas que utilizam como base um Modelo Lógico desenvolvido com foco na representação de dispositivos adaptativos dirigidos por regras. Desta forma, facilita para que um especialista em dispositivos adaptativos possa estender um determinado dispositivo não adaptativo dirigido por regras em adaptativo. Sendo assim, ao final do projeto, busca-se obter uma ferramenta que permitirá auxiliar um especialista em seu projeto.**

***Keywords*—adaptativa, tecnologia adaptativa, framework adaptativo, simulação adaptativa**

1. INTRODUÇÃO

APLICAÇÕES complexas são caracterizadas por componentes e aspectos cuja estrutura e comportamento, comumente, podem modificar-se [1]. Tais aplicações possuem um comportamento inicial definido por um conjunto de ações que desempenham suas funções elementares e podem ter o seu comportamento modificado durante a execução para dar suporte a novas funcionalidades. Tais modificações são decorrentes dos estímulos de entrada a que são submetidos no sistema e/ou da ocorrência de suas ações internas.

Uma técnica utilizada para auxiliar os projetistas no projeto de aplicações com comportamento modificável é a tecnologia adaptativa [2]. A tecnologia adaptativa envolve um dispositivo não-adaptativo (subjacente) já existente em uma camada adaptativa que permite realizar mudanças no comportamento da aplicação definida [3]. É possível citar, por exemplo, trabalhos relacionados a reconhecedores sintáticos adaptativos [4], os *Statecharts* Adaptativos [5] - empregados na modelagem de sistemas reativos - e a modelagem de aplicações complexas com base no ISDL Adaptativo [6]. O desenvolvimento de tecnologia adaptativa aplicado a sistemas de dispositivos não adaptativos dirigidos por regras vem sendo pesquisada com totais preocupações a fim de que o usuário consiga gerenciar novos dispositivos adaptativos.

Camolesi propôs em [1] um gerador de ambientes (metambiente) que possibilita a geração automática de ambientes para o projeto de aplicações adaptativas. Tal gerador fundamenta-se nos conceitos de Tecnologia Adaptativa e permite a definição de dispositivos adaptativos dirigidos por regras [7].

No trabalho apresentado em [1] uma das etapas necessárias para o desenvolvimento do gerador de ambientes
é a construção de um *framework* para auxiliar na especificação de formas de operação para dispositivos adaptativos. Tal *framework* deve representar os elementos conceituais de operação de um dispositivo e permitir realizar a simulação e a verificação de aplicações especificadas com base em um formalismo adaptativo representado.

A. Problema

As ferramentas existentes para o projeto de aplicações são muito especificas e restritas a um ou alguns formalismos apenas e não contemplam o uso de dispositivos adaptativos. Quando um novo dispositivo adaptativo é projetado, novas ferramentas para especificação, simulação e verificação de aplicações devem ser construídas. O desenvolvimento de tais ferramentas é lento e acaba desestimulando a criação de novos dispositivos adaptativos. Este projeto teve o propósito de disponibilizar um framework para a definição da forma de operação de dispositivos adaptativos. O *framework* fornece aos especialistas em dispositivos adaptativos uma ferramenta para auxiliar na definição de novos dispositivos adaptativos de forma que ao especificarem um novo dispositivo já obtenham de forma automática ferramentas para a simulação e a verificação de aplicações produzidas com base no novo formalismo gerado.

B. Objetivos

Desenvolver e disponibilizar um *framework* para definição de forma de operação para dispositivos adaptativos. Tal *framework* possibilita depois de realizada à definição da forma de operação de um dispositivo a simulação e a verificação de aplicações especificadas com base no dispositivo adaptativo obtido.

C. Relevância

Com a concretização desse trabalho obteve-se uma ferramenta que permite a um especialista em certo dispositivo dirigido por regra realizar o mapeamento deste para uma metaestrutura que represente sua extensão adaptativa e a sua forma de operação. Além do auxílio aos especialistas em extensão de dispositivos adaptativos, a ferramenta também auxilia os projetistas na realização do trabalho de especificação e simulação de suas aplicações.

D. Estrutura do trabalho

Este trabalho encontra-se organizado da seguinte forma: inicialmente, no Capítulo 1 foi apresentada uma visão geral do trabalho, seus objetivos e as suas justificativas. Na sequência, o Capítulo 2 apresenta uma breve introdução dos conceitos de Tecnologia Adaptativa, de ferramentas para o projeto de aplicações que se utilizam deste conceito e descreve a arquitetura geral do Modelo Lógico utilizado para suportar a representação computacional de dispositivos adaptativos. Como base no Modelo Lógico foi realizado o desenvolvimento do Framework Adaptativo, descrito no

Capítulo 3. Por fim, no Capítulo 4 serão tecidas algumas conclusões e trabalhos futuros.

2. TECNOLOGIA ADAPTATIVA

O trabalho proposto fundamenta-se na Tecnologia Adaptativa, a qual permite que um dispositivo adaptativo seja capaz de se auto modificar, ou seja, seu comportamento mudar dinamicamente em tempo de execução e sem influência externa, isso quando ele detecta uma situação em que para ele não é mais possível dar continuidade a sua execução sem antes ele alterar seu comportamento.

A maior vantagem dos dispositivos adaptativos é a sua facilidade de uso, sua relativa simplicidade e o fato de ser muito rara a necessidade de existir uma descrição completamente especificada do comportamento do dispositivo para que ele possa ser utilizado. Em lugar disso, sua operação pode ser descrita de forma incremental, e seu comportamento, programado para se alterar dinamicamente em resposta aos estímulos de entrada recebidos. Adicionalmente, como o conjunto de regras que definem o seu comportamento também se altera ao longo da sua operação, pode-se dizer que o próprio dispositivo adaptativo programa de alguma forma a representação do conhecimento adquirido através de sua sequencia recebida de estímulos de entrada.

Dispositivos adaptativos podem ser empregados em uma grande variedade de situações, principalmente em segmentos complexos da solução de um problema, no qual sejam realizadas tomadas não triviais de decisão.

Para auxiliar o projeto de aplicações que use tecnologia adaptativa, faz-se necessária a utilização de um ambiente que integra um conjunto de ferramentas que dê suporte aos projetistas na realização de seu trabalho. A Fig. 1 [1] ilustra a arquitetura geral de tal ambiente, que é organizado em 3 (três) grupos de ferramentas: Ferramentas de Edição, Ferramentas de Análise e Ferramenta de Implementação.

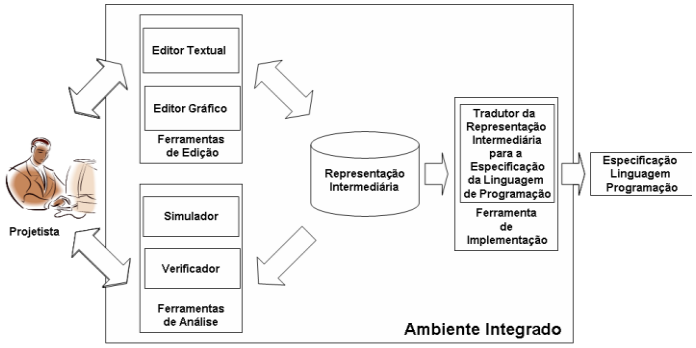

Figura 1. Ambiente para o projeto de aplicações usando Tecnologia Adaptativa.

Valendo-se das Ferramentas de Edição, um projetista de aplicações utilizando-se de um dispositivo adaptativo específico poderá realizar a especificação de sua aplicação com o auxilio de um editor de texto qualquer ou de um editor gráfico. Para possibilitar o intercâmbio das especificações produzidas, os editores devem gerar objetos no Modelo Lógico. Caso a especificação seja produzida em um editor textual, esta deverá ser compilada para transformar a codificação realizada no formato definido para o modelo lógico. Depois de realizada a especificação, o projetista de aplicações poderá utilizar as Ferramentas de Análise. Tais ferramentas utilizam a codificação da especificação (no formato do modelo lógico) como base e permitem a realização de uma análise do comportamento da aplicação adaptativa em desenvolvimento. Por fim, depois de especificada e analisada a representação de uma aplicação, o projetista pode se utilizar das Ferramentas de Produção de Representações Físicas e gerar uma representação de uma aplicação em um determinado padrão de linguagem de representação física e obter a aplicação desejada.

A. Modelo Lógico para Ambiente Adaptativo

O Modelo Lógico proposto em [1] é o núcleo central do Framework Adaptativo proposto. Neste contexto será descrito neste capítulo um resumo do modelo lógico para representação de dispositivos adaptativos e aplicações modeladas com base nestes dispositivos. O Modelo Lógico é dividido em quatro camadas:

- Camada de especificação de dispositivos;
- Camada de especificação de aplicações – não adaptativa;
- Camada de especificação de funções e ações adaptativas;
- Camada de representação de memória;

A definição do modelo em camada é fundamental para o desenvolvimento do trabalho, uma vez que permite separar a camada não adaptativa da camada adaptativa. Abaixo (Fig. 2) será mostrado o modelo lógico (diagrama de classe) completo.

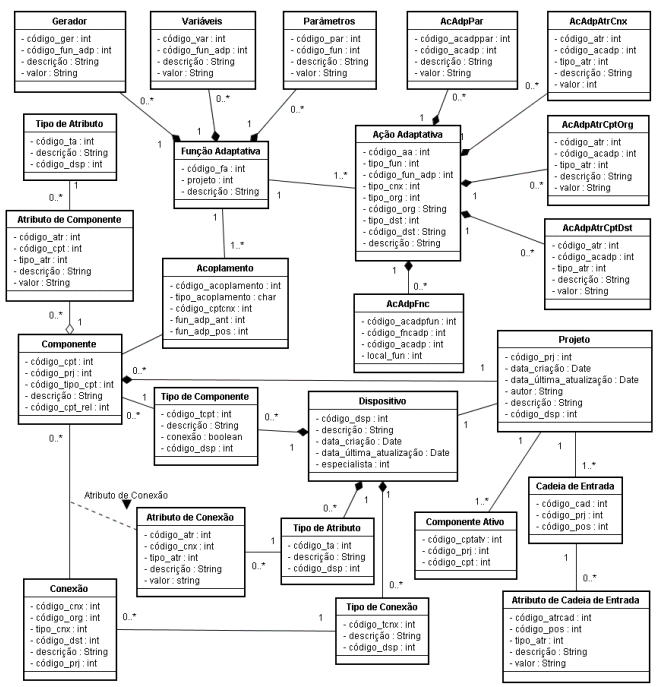

Figura 2. Modelo Lógico (Camolesi, 2007)

Conforme dito anteriormente o Modelo Lógico foi estruturado em camadas. A seguir serão apresentados os elementos que constituem cada camada e os seus respectivos

relacionamentos.

A1) Camada de Especificação de Dispositivos

Os elementos da Camada de Configuração de Dispositivos, também denominada na teoria, como Núcleo Subjacente [7], representa os elementos lógicos referentes à definição de um determinado dispositivo dirigido por regras não adaptativo. A Fig. 3 ilustra esta camada, na qual um especialista depois de estender os conceitos formais de um determinado dispositivo deve realizar um mapeamento da estrutura conceitual do referido dispositivo para a mesma. Ao instanciar objetos nesta estrutura o especialista define os elementos conceituais: tipos de componentes, tipos de conexões e tipos de atributos que definem a estrutura de dispositivo dirigido por regras não adaptativos.

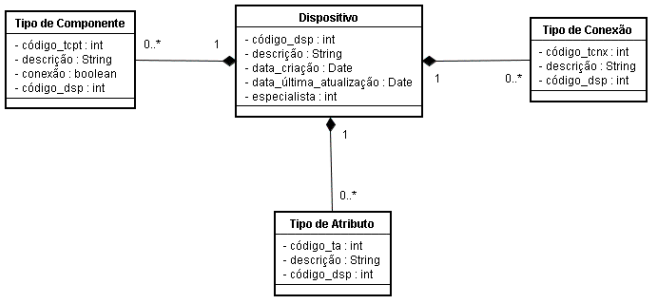

Figura 3. Camada de Configuração de Dispositivos.

A2) Camada de Especificação de Aplicações

A Camada de Especificação de Aplicações – Não adaptativa tem por objetivo representar os elementos referentes ao comportamento de uma determinada aplicação que está sendo modelada por certo projetista. Um projetista ao modelar a aplicação instancia objetos da camada de núcleo subjacente e define o comportamento da aplicação gerando novos objetos na camada de especificação de aplicações. Para cada elemento definido nesta camada são associados os seus tipos correspondentes à camada de núcleo subjacente e também é feita uma relação entre os elementos dos diversos conjuntos que representam esta camada. Esta camada, como foi dito anteriormente, representa o comportamento da aplicação e, consequentemente, a especificação não adaptativa de uma determinada aplicação. A Fig. 4 apresenta a organização estrutural da referida camada.

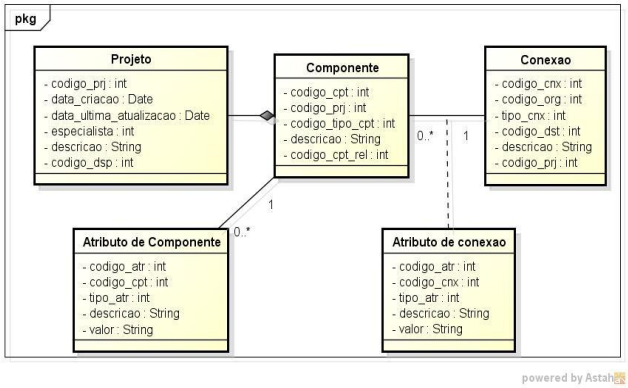

Figura 4. Camada de Especificação de Aplicações – Subjacente.

A3) Camada de Especificação de Funções e Ações Adaptativa

A Camada de Especificações de Funções e Ações Adaptativas tem por objetivo armazenar informações referentes às funções e ações adaptativas que são utilizadas em uma determinada aplicação. Os objetos desta camada são definidos com base nos objetos da camada de especificação, pois deve estar associado aos componentes e conexões de um determinado projeto. A utilização desta camada é de fundamental importância para a estrutura proposta neste trabalho.

Esta camada representa os elementos conceituais definidos em [7] para a concepção de um dispositivo adaptativo. Desta forma uma vez que um determinado projetista define um novo dispositivo na Camada de Configuração de Dispositivos (Fig. 5), estará desta forma estendendo o referido dispositivo para suportar tecnologia adaptativa.

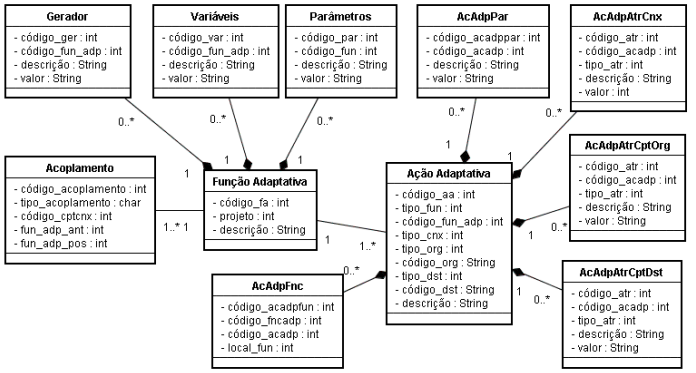

Figura 5. Camada de Especificação de Funções e Ações Adaptativa.

A4) Camada de Representação de Memória

Por fim, a Camada de Memória (Fig. 6) representa os estímulos de entrada (cadeia de entrada, estímulos externos oriundos de outros agentes, etc..) e o status (estados ou transições habilitadas a ocorrerem). Esta camada é de fundamental importância para a construção de ferramentas que darão suporte a simulações e verificações de aplicações projetadas utilizando-se de tecnologia adaptativa.

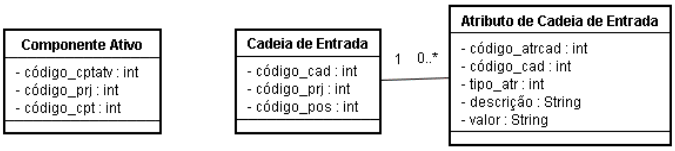

Figura 6. Camada de Representação de Memória.

3) Framework para Simulações de Tecnologias Adaptativas

Com o foco no estudo e aplicação dos conceitos de tecnologias adaptativas foi definido um estudo de caso que consiste no projeto e no desenvolvimento de um *framework* para geração de simulação de funções adaptativas. O *framework* concebido tem por objetivo de utilizar as definições no Modelo Lógico proposto. Tal framework tem por objetivo auxiliar os projetistas nas simulações de suas

aplicações. Um especialista ao iniciar uma nova simulação ganhará conhecimento suficiente para continuar com seu projeto, gerando grande base de conhecimento do funcionamento de sua aplicação em várias situações. Com isto obtém-se uma grande economia de tempo e dinheiro na concepção de suas tarefas.

A) Projeto do Framework para Simulações de Tecnologias Adaptativas

O *framework* proposto foi estruturado de forma a facilitar os estudos sobre tecnologias adaptativas, a partir do modelo lógico já existente. O *framework* gera uma simulação de um dispositivo mapeado pelo especialista com as definições básicas de seu projeto, porem ele terá de colocar as funções adaptativas com as informações necessárias para que permita simular diversos tipos de comportamento previsto. O framework foi desenvolvido com as regras de orientação a objetos, ou seja, foi estruturado em camadas.

Com base nos requisitos mínimos definidos para a concepção do framework o mesmo foi organizado em seis etapas que permitem desde a seleção do dispositivo ate a execução da simulação. As etapas são encadeadas com as informações pré-mapeadas pelo especialista. A Fig. 7 ilustra as etapas e o funcionamento do *framework* descrito.

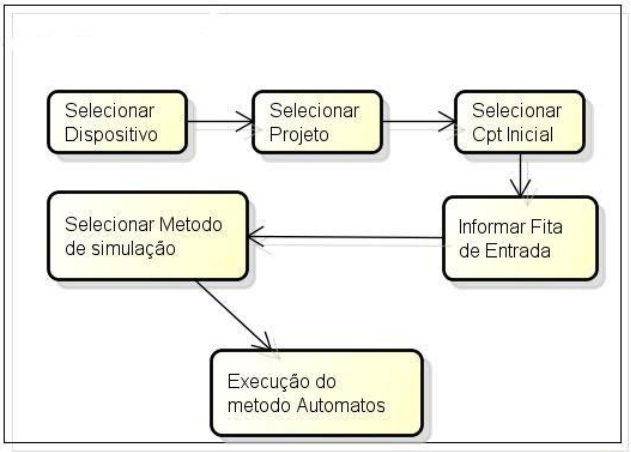

Figura 7. Etapas do Framework para Simulação Adaptativa.

Conforme apresentado na Fig. 7 o framework é definido em seis etapas, sendo:

- **Seleção de dispositivo**: etapa responsável por separar as informações desejadas para a simulação;
- **Seleção de projeto**: após a seleção do dispositivo, o framework ira mostrar os projetos relacionados ao dispositivo selecionado;
- **Seleção de componente inicial**: permite a seleção do componente inicial onde se iniciara a simulação do dispositivo;
- **Alimentação da fita de entrada**: A fita de entrada é a condição em que ocorrera a simulação do dispositivo, podendo ser informada a condição em que se quer testar o dispositivo selecionado;
- **Seleção do método de simulação**: Nesta etapa será apresentado ao especialista duas opções de execução, a primeira realiza toda a simulação sem pausas, executando a simulação do dispositivo e só mostrando o resultado final obtido. A segunda opção realiza a simulação com pausas. Neste tipo de simulação o framework mostrara cada etapa do processo e o estado da aplicação no momento da pausa.
- **Execução do método autômatos**: Neste local é onde ocorre a simulação do dispositivo selecionado com as informações da fita de entrada.

B) Implementação do framework

Na etapa inicial foram levantados os dados para a criação do framework, com base em alguns dispositivos já existentes, foi escolhido o autômato de estado finito determinístico adaptativo. A Figura 8 apresenta a interface do framework gerado para a simulação do dispositivo mencionado acima.

Figura 8. Interface para simulação de autômatos de estados finitos determinística.

No trecho de , apresentado abaixo, é responsável por carregar os dados dos dispositivos e os seus respectivos projetos. Neste local pode-se selecionar um dispositivo e os seus respectivos projetos existentes.

Código 1. Trecho de programa para seleção do dispositivo e projetos**.**

```
private void  cmbDsp_SelectedValueChanged(object sender, EventArgs e)
      {
         lbnDspId.Text = cmbDsp.SelectedValue.ToString();
         try            idDsp = int.Parse(lbnDspId.Text);
            oDsp = bllDsp.SelectCod(idDsp);
            lbnDspId.Text = oDsp.codigo_dsp.ToString();
            lbnDspDsc.Text = oDsp.descricao.ToString();
            lbnDspNome.Text = oDsp.especialista.ToString();
            dgvTpCpt.DataSource = bllDsp.SelectCod(idDsp).tipo_de_
               componente;
            dgvTpCnx.DataSource = bllDsp.SelectCod(idDsp).tipo_de_conexao;
            dgvTpAtr.DataSource = bllDsp.SelectCod(idDsp).tipo_de_atributo;
            cmbPrj.DataSource = bllPrj.SelectCodDsp(idDsp);
      }
private void cmbPrj_SelectedValueChanged(object sender, EventArgs e)
      {
         lbnPrjId.Text = cmbPrj.SelectedValue.ToString();
         try
         {
            idPrj = int.Parse(lbnPrjId.Text);
            oPrj = bllPrj.SelectCod(idPrj);
            lbnPrjId.Text = oPrj.codigo_prj.ToString();
            lbnDescPrj.Text = oPrj.descricao.ToString();
            lbnNomePrj.Text = oPrj.autor.ToString();
            dgvCnx.DataSource = bllPrj.SelectCod(idPrj).conexao;
            dgvCpt.DataSource = bllCpt.SelectCodPrj(idPrj);
            dgvCptIni.DataSource = bllPrj.SelectCod(idPrj).componente1;
            dgvFuncAdp.DataSource = bllFncAdp.SelectCodPrj(idPrj);
         }
```

Em seguida, após a seleção do dispositivo e a escolha de um mapeado anteriormente são carregados todos os componentes, atributos de componentes, conexões e atributos de conexões do respectivo projeto selecionado com as configurações do projeto escolhido pelo especialista.

Após esta etapa é apresentado na interface do framework uma janela com opções para o especialista definir qual o tipo de simulação (***Pausa*** com paradas para o especialista avaliar o estado da aplicação ou ***Contínua*** que realiza toda a simulação apresentando apenas a situação final) ele deseja (Fig. 9).

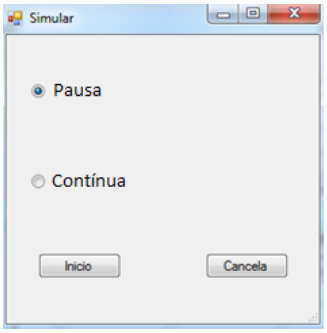

Figura 9. Interface para escolha do tipo de simulação**.**

Depois de escolhidas uma das opções (*Pausa* ou *Contínua*) e pressionado o botão "Início" ocorre a execução do trecho de código Código 2. O referido trecho é responsável por iniciar a simulação. Inicialmente são carregadas as informações escolhida pelo projetista e, na sequencia, é fechada a janela de escolha de tipo de simulação, iniciando a mesma.

Código 2. Método btnCancel_Click – Menu Simular**.**

```
private void btnCancel_Click(object sender, EventArgs e)
    {
       this.Close();
    }

    private void btnInicio_Click(object sender, EventArgs e)
    {
       if (semParada.Checked)
       {
          sele = 1;
       }
       else {
          if (comTemp.Checked)
          {
             sele = 2;
          }
          else { sele = 3; }
       }
       this.Close();
    }
```

Com esta etapa concluída, o framework reúne as informações que foram definidas pelo especialista (modelo da aplicação) para começar a simulação do comportamento da mesma. Tais informações serão armazenadas na classe autômato contendo identificação do componente inicial (idCpt), fita de entrada (fita), método de simulação (selecionado) e identificação do projeto (idPrj).

Com a primeira etapa concluída, o framework começará a simulação da aplicação modelada, desta forma ele executa as atividades de acordo com a execução definida no diagrama apresentado na Fig. 4.

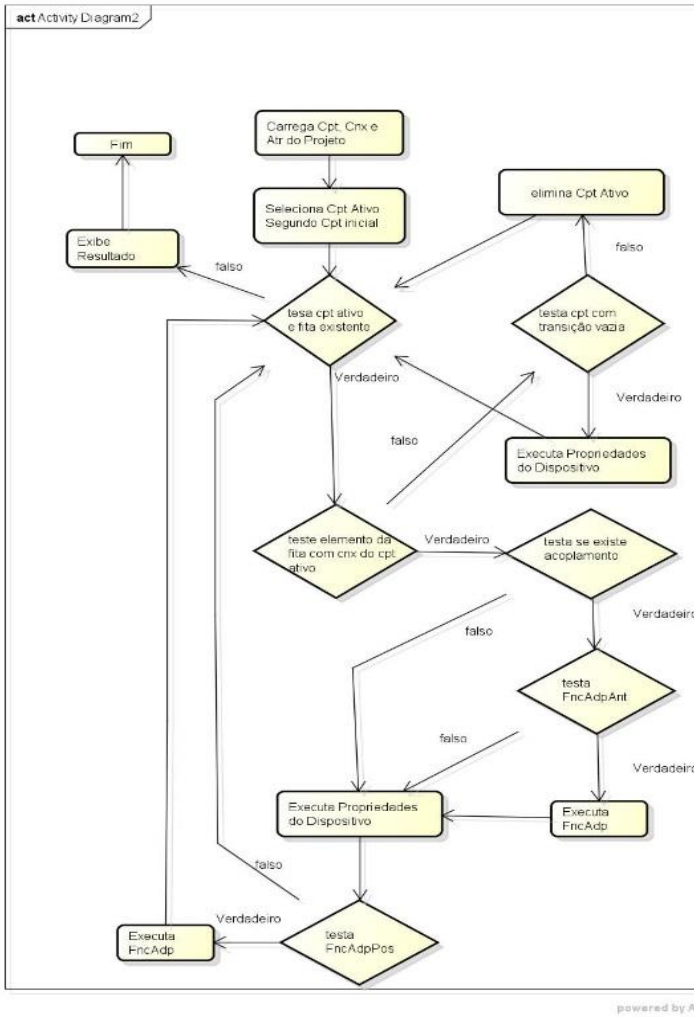

Figura 10. Diagrama de atividades do método automatos**.**

O diagrama descrito acima se refere aos trechos de código de 3 a 7, no trecho de Código 3 o método ***automatos*** carrega todos os componentes do dispositivo junto com suas conexões e atributos, além de posicionar a fita de entrada na posição inicial e selecionar o primeiro componente ativo.

Código **3**. Método automatos - Parte 1.

```
public void automatos()
    {
       par.Teste = "";
       idCptAux = idCpt;
       cont1 = cont2 = ca = 0;
       lCpt = new List<componente>();
       lCpt = cptDal.SelectCodPrj(idPrj);
       if (lCpt.Count != 0)
       {   for (int i = 0; i < lCpt.Count; i++)
             if (lCpt[i].codigo_cpt == idCptAux)
             {
                CptAtv = new componente_ativo();
                CptAtv.codigo_prj = idPrj;
                CptAtv.codigo_cpt = lCpt[i].codigo_cpt;
                oCpt = lCpt[i];

        }     }
       lCnx = new List<conexao>();
       lAtrCnx = new List<atributo_de_conexao>();
       lCnx = cnxDal.SelectCodPrj(CptAtv.codigo_prj.Value);
       for (int i = 0; i < lCnx.Count; i++)
       {  lAtrCnxAux = new List<atributo_de_conexao>();
          lAtrCnxAux = atrCnxDal.SelectCodCnx
             (lCnx[i].codigo_cnx);
          for (int j = 0; j < lAtrCnxAux.Count; j++)
             lAtrCnx.Add(lAtrCnxAux[j]);
```

O Código 4, ilustra a parte 2, do método automatos, na qual o comando "*while*" realiza uma iteração responsável por fazer a execução das ações necessários para que o dispositivo execute todas as suas tarefas. No fim do *while* após o teste é verificado para a posição da fita com as conexões do dispositivo ativo se existe uma transição para o estado corrente com o respectivo símbolo selecionado.

Código 4. Método automatos - Parte 2.

```
while ((CptAtv != null) && (fita.Length !=0))
        {  par.BusAtrCnx = lAtrCnx;
           par.LCpt = lCpt;
           par.Lcnx = lCnx;
           lCnxAux = new List<conexao>();
           for (int i = 0; i < lCnx.Count; i++)
           {   if (lCnx[i].codigo_org == CptAtv.codigo_cpt) {
               lCnxAux.Add(lCnx[i]);
            }}
           lAtrCnxAux = new List<atributo_de_conexao>();
           for (int i = 0; i < lCnxAux.Count; i++) {
              for (int j = 0; j < lAtrCnx.Count; j++) {
                 if (lCnxAux[i].codigo_cnx == lAtrCnx[j].codigo_cnx) {
                    lAtrCnxAux.Add(lAtrCnx[j]);
            } }              }
           menuSeleciona();
           bool teste = false;
           atrCnx = new atributo_de_conexao();
           if (fita.Length > ca)
           {for (int i = 0; i < lAtrCnxAux.Count; i++)
              { if (lAtrCnxAux[i].valor.Equals
              ((fita.ElementAt(ca)).ToString()))
                 {  teste = true;
                    atrCnx = lAtrCnxAux[i];
            }} }
```

Caso o teste de verificação de transição com símbolo vigente seja verdadeiro, o trecho de Código 5 é executado. Este trecho tem a função de verificar se existe algum acoplamento para a conexão ativa, caso exista um acoplamento, o código testa se existe alguma função adaptativa anterior para ser executada. Na existência da função adaptativa anterior, está é executada e a seguir o método executa os passos necessários para o dispositivo descrito anteriormente e, por fim, carrega o novo componente ativo para o próximo ciclo.

Código 5. Método automatos - Parte 3.

```
if (teste)
          {
            lAcp = new List<acoplamento>();
            lAcp = acpDal.SelectCodCpt(atrCnx.codigo_cnx.Value);
            oAcp = new acoplamento();
            if (lAcp.Count != 0)
               oAcp = lAcp[0];
            if ((oAcp != null) && (oAcp.fun_adp_ant != 0)
            && (oAcp.fun_adp_ant != null))
               funcAdp(oAcp.fun_adp_ant.Value);
            ca++;
            CptAtv = null;
            for (int i = 0; i < lCnxAux.Count; i++)
            {
               if (lCnxAux[i].codigo_cnx == atrCnx.codigo_cnx)
               {  CptAtv = new componente_ativo();
                  CptAtv.codigo_cpt = lCnxAux[i].codigo_dst;
                  CptAtv.codigo_prj = idPrj;
                  for (int j = 0; j < lCpt.Count; j++)
                  { if (lCpt[j].codigo_cpt == CptAtv.codigo_cpt)
                        oCpt = lCpt[j];
                  }
               }
            }
```

O trecho de Código 6 ilustra o teste do acoplamento para função adaptativa posterior que ocorre após a execução das tarefas elementares do dispositivo. Na existência de uma função adaptativa posterior, ela é executada. No caso da busca de conexões com símbolo corrente resultar falso, ou seja, não existem conexões com símbolo definido para o respectivo estado, é realizado o teste para verificação das transições espontâneas "ε". Caso também não exista transições vazias o método sai do *while* e finaliza a execução da simulação, retornando erro.

Código 6. Método automatos - Parte 4.

```
if ((oAcp != null) && (oAcp.fun_adp_pos != 0)
&& (oAcp.fun_adp_pos != null))
            funcAdp(oAcp.fun_adp_pos.Value);
            else {
               for (int i = 0; i < lAtrCnxAux.Count; i++)
                  if (lAtrCnxAux[i].valor.Equals("e"))
                  {  teste = true;
                     atrCnx = lAtrCnxAux[i];
                  }
               if (teste)
                  CptAtv = null;
                  for (int i = 0; i < lCnxAux.Count; i++)
                     if (lCnxAux[i].codigo_cnx == atrCnx.codigo_cnx)
                     {
                        CptAtv = new componente_ativo();
                        CptAtv.codigo_cpt = lCnxAux[i].codigo_dst;
                        CptAtv.codigo_prj = idPrj;
                        for (int j = 0; j < lCpt.Count; j++)
                          if (lCpt[j].codigo_cpt == CptAtv.codigo_cpt)
                             oCpt = lCpt[j];
               }
```

No caso do trecho de código abaixo, a função testa a fita de entrada para verificar se foi feito todas os passos para o dispositivo junto com o teste para verificar se o componente é de estado final. Caso todos os testes sejam satisfeitos a simulação "Aceita" a cadeia de entrada, caso contrário, a cadeia de entrada é "Rejeita" e ocorre o fim da simulação.

Código 7. Método automatos - Parte 5.

```
else {
                     CptAtv = null;
                  }

               }
            }
            if (ca == (fita.Length) && oCpt.codigo_tipo_cpt == 15)
            {
               par.Teste = "Aceita";
            }
            else {
               par.Teste = "Rejeita";
            }
            par.ShowDialog();

         }
```

A seguir é representado o trecho de código da função adaptativa, que pode ser chamada ate duas vezes no trecho de Código 4 (Função Adaptativa Anterior e Função Adaptativa Posterior). Incialmente o código testa a função para selecionar e separa todas as ações adaptativas relacionadas a função em execução, após a seleção, o código testa qual tipo de ação que será executada.

Código 8. Método funcAdp.

```
public void funcAdp(int cod) {
        funAdp = new funcao_adaptativa();
        lacAdp = new List<acao_adaptativa>();
        funAdp = funAdpDal.SelectCod(cod);
        lacAdp = acAdpDal.SelectCodFncAdp(funAdp.codigo_fa);
        int i = 0;
        while (i < lacAdp.Count)
        {  int caseSwitch = lacAdp[i].tipo_fun.Value;
           switch (caseSwitch)
           {
              case 1:
                 busca(lacAdp[i].codigo_aa);
                 break;
              case 2:
                 remoção();
                 break;
              case 3:
                 adicao(lacAdp[i]);
                 break;
           }
        i++;
}       }
```

Os conceitos de tecnologia adaptativa apresentados por Neto, 2001 e mapeados para o modelo lógico de Camolesi, 2007 foram utilizados para o desenvolvimento da execução de cada ação adaptativa. Sendo assim, cada ação adaptativa possui um atributo que indica o tipo de ação que vai ser executado: 1 - que representam a busca, 2 - remoção e 3 - adição.

Na ocorrência de uma ação de busca, o framework procura pela conexão compatível com a conexão da função adaptativa que foi mapeada junto com o dispositivo. Se a ação for de remoção, o framework removera a conexão selecionada na busca, e caso a ação for de adição, o framework será responsável por adicionar novas regras a aplicação que está sendo simulada. O Código 9, apresentado a seguir, ilustra a execução das ações adaptativas representadas no método busca.

Código 9. Método busca.

```
public void busca(int cod) {
        lbusAtrCnx = new List<atributo_de_conexao>();
        latraa = new List<AcAdpAtrCnx>();
        latraa = atraaDal.SelectCodAcAdp(cod);
        for(int i=0;i<latraa.Count;i++){
           for (int j = 0; j < lAtrCnx.Count; j++)
           {
              if (latraa[i].valor == lAtrCnx[j].valor)
              {
                 lbusAtrCnx.Add(lAtrCnx[j]);
              }
           }
        }
        lbusCnx = new List<conexao>();
        for (int i = 0; i < lCnx.Count; i++) {
           for (int j = 0; j < lbusAtrCnx.Count; j++) {
              if (lCnx[i].codigo_cnx == lbusAtrCnx[j].codigo_cnx) {
                 lbusCnx.Add(lCnx[i]);
              }
           }
        }
     }
```

O trecho de Código 10 ilustra a funcionalidade definida para as ações adaptativas de remoção. A execução desta ação de remoção utiliza a conexão encontrada pela ação de busca e a remove das conexões existentes referentes ao projeto selecionado. Além disso, é armazenado os códigos de identificação dos componentes que a conexão fazia parte.

Código 10. Método remoção.

```
public void remoção()
     {
        for (int i = 0; i < lCnx.Count; i++)
        {
           for (int j = 0; j < lbusCnx.Count; j++)
           {
              if (lCnx[i].codigo_cnx == lbusCnx[j].codigo_cnx)
              {
                 var1 = new variaveis();
                 var2 = new variaveis();
                 var1.descricao = lCnx[i].codigo_org.ToString();
                 var2.descricao = lCnx[i].codigo_dst.ToString();
                 lCnx.RemoveAt(i);
              }
           }
        }
        for (int i = 0; i < lAtrCnx.Count; i++)
        {
           for (int j = 0; j < lbusAtrCnx.Count; j++)
           {
              if (lAtrCnx[i].codigo_atr == lbusAtrCnx[j].codigo_atr)
              {
                 lAtrCnx.RemoveAt(i);
              }
           }
        }

     }
```

A ação de remoção só ira apagar o que foi encontrado pela ação de busca, já a função de adição tem algumas condições, por exemplo: caso exista algum argumento dela com "*" e logo após "?", a função ira criar um componente e em seguida criar a conexão necessária para que este componente seja utilizado, no Código 11, inicialmente, a ação de adição faz um teste para saber se vai utilizar o código do componente anterior para fazer a conexão do código do componente que será criado pelo código. A criação de uma nova conexão segue a seguinte ordem, componente, conexão e atributo de conexão.

Código 11. Método adicao - Parte 1.

```
public void adicao(acao_adaptativa acAdp) {
        if (acAdp.codigo_org.Contains("?") &&
acAdp.codigo_dst.Contains("*"))
        {
           cont1++;
           componente cptAux = new componente();
           cptAux.codigo_cpt = lCpt.Last().codigo_cpt + 1;
           cptAux.codigo_prj = idPrj;
           cptAux.codigo_tipo_cpt = 16;
           cptAux.descricao = "k" + cont1;
           cptAux.codigo_cpt_rel = 0;
           lCpt.Add(cptAux);
           conexao cnxAux = new conexao();
           cnxAux.codigo_cnx = lCnx.Last().codigo_cnx + 1;
           cnxAux.codigo_org = int.Parse(var1.descricao);
           cnxAux.tipo_cnx = 9;
           cnxAux.codigo_dst = cptAux.codigo_cpt;
           cnxAux.codigo_prj = idPrj;
           cnxAux.descricao = acAdp.descricao;
           lCnx.Add(cnxAux);
           atributo_de_conexao atrAux = new atributo_de_conexao();
           AcAdpAtrCnx atraaAux = new AcAdpAtrCnx();
           latraa = new List<AcAdpAtrCnx>();
           latraa = atraaDal.SelectCodAcAdp(acAdp.codigo_aa);
           for (int i = 0; i < latraa.Count; i++)
           {
              if (latraa[i].codigo_acadp == acAdp.codigo_aa)
              {
                 atraaAux = latraa[i];
              }
           }
```

O Código 12 será executado quando a função adaptativa tem que criar um componente e criar a conexão com um componente existente, a ordem da criação e a mesma do código anterior.

Código 12. Método adicao - Parte 2.

```
atrAux.codigo_atr = lAtrCnx.Last().codigo_atr + 1;
            atrAux.codigo_cnx = cnxAux.codigo_cnx;
            atrAux.tipo_atr = 9;
            atrAux.descricao = atraaAux.descricao;
            atrAux.valor = atraaAux.valor;
            lAtrCnx.Add(atrAux);
            var1.descricao = cptAux.codigo_cpt.ToString();
         }
         else {
            if (acAdp.codigo_org.Contains("*") &&
acAdp.codigo_dst.Contains("?"))
            {  cont2++;
               componente cptAux = new componente();
               cptAux.codigo_cpt = lCpt.Last().codigo_cpt + 1;
               cptAux.codigo_prj = idPrj;
               cptAux.codigo_tipo_cpt = 16;
               cptAux.descricao = "k1 " + cont2;
               cptAux.codigo_cpt_rel = 0;
               lCpt.Add(cptAux);
               conexao cnxAux = new conexao();
               cnxAux.codigo_cnx = lCnx.Last().codigo_cnx + 1;
               cnxAux.codigo_dst = int.Parse(var2.descricao);
               cnxAux.tipo_cnx = 9;
               cnxAux.codigo_org = cptAux.codigo_cpt;
               cnxAux.codigo_prj = idPrj;
               cnxAux.descricao = acAdp.descricao;
               lCnx.Add(cnxAux);
               atributo_de_conexao atrAux = new atributo_de_conexao();
               AcAdpAtrCnx atraaAux = new AcAdpAtrCnx();
               latraa = new List<AcAdpAtrCnx>();
               latraa = atraaDal.SelectCodAcAdp(acAdp.codigo_aa);
```

Caso o teste retorne vazio, será criada uma conexão com transição vazia, já explicada anteriormente.

Código 13. Metodo adicao - Parte 3.

```
for (int i = 0; i < latraa.Count; i++)
            {  if (latraa[i].codigo_acadp == acAdp.codigo_aa)
                     atraaAux = latraa[i];
               atrAux.codigo_atr = lAtrCnx.Last().codigo_atr + 1;
               atrAux.codigo_cnx = cnxAux.codigo_cnx;
               atrAux.tipo_atr = 9;
               atrAux.descricao = atraaAux.descricao;
               atrAux.valor = atraaAux.valor;
               lAtrCnx.Add(atrAux);
               var2.descricao = cptAux.codigo_cpt.ToString();
            }
            else {
               conexao cnxAux = new conexao();
               cnxAux.codigo_cnx = lCnx.Last().codigo_cnx + 1;
               cnxAux.codigo_org = int.Parse(var1.descricao);
               cnxAux.tipo_cnx = 9;
               cnxAux.codigo_dst = int.Parse(var2.descricao);
               cnxAux.codigo_prj = idPrj;
               cnxAux.descricao = acAdp.descricao;
               lCnx.Add(cnxAux);
               atributo_de_conexao atrAux = new
atributo_de_conexao();
               AcAdpAtrCnx atraaAux = new AcAdpAtrCnx();
               latraa = new List<AcAdpAtrCnx>();
               latraa = atraaDal.SelectCodAcAdp(acAdp.codigo_aa);
               for (int i = 0; i < latraa.Count; i++)
                  if (latraa[i].codigo_acadp == acAdp.codigo_aa)
                     atraaAux = latraa[i];
               atrAux.codigo_atr = lAtrCnx.Last().codigo_atr + 1;
               atrAux.codigo_cnx = cnxAux.codigo_cnx;
               atrAux.tipo_atr = 9;
               atrAux.descricao = atraaAux.descricao;
               atrAux.valor = atraaAux.valor;
               lAtrCnx.Add(atrAux);
            }} }
```

4) CONCLUSÃO

Neste trabalho foram apresentados os conceitos de Tecnologia Adaptativa, de um modelo lógico para representação para dispositivos adaptativos definidos por regras e a construção de um framework para auxiliar a construção de ferramentas para simulação de aplicações definidas com dispositivos adaptativos.

O trabalho realizado permitiu validar o modelo lógico apresentado em Camolesi, 2007. Tal modelo demonstrou-se compatível com os conceitos teóricos dos dispositivos subjacentes estudados e permitiu verificar que o modelo lógico também é consistente para a representação computacional de aplicações adaptativas modeladas com base em dispositivos adaptativos dirigidos por regras.

O framework desenvolvido foi instanciado para a representação de autômatos finitos adaptativos e demonstrou-se eficiente para a sua realização. Com base na ferramenta de simulação obtida foram realizados testes e verificou-se que tal ferramenta pode auxiliar os projetistas de aplicações no desenvolvimento de suas tarefas.

A arquitetura do framework desenvolvido permite que um especialista em programação e em um determinado dispositivo adaptativo obtenha de forma rápida uma ferramenta para simulação de aplicações.

Fundamentado na pesquisa desenvolvida, várias vertentes para trabalhos futuros podem ser identificadas. Como possíveis trabalhos futuros, pode-se citar:

- Utilização do framework desenvolvido para instanciação de outros dispositivos adaptativos, com o intuito de avaliar a consistência da implementação realizada;
- Desenvolvimento de uma ferramenta textual, acoplada ao framework para auxiliar a definição de novos dispositivos adaptativos e o projeto de aplicações com base nos dispositivos obtidos;
- Desenvolvimento de uma ferramenta gráfica para auxiliar o projeto de aplicações fundamentadas em dispositivos adaptativos;
- Melhorias no framework para definição de uma linguagem que permita a um leigo em programação definir por meio de uma ferramenta textual o comportamento de um dispositivo adaptativo, sem ter necessidade de utilizar-se de recursos de programação, ou seja, realizar a implementação de uma máquina virtual adaptativa.
- Construção de uma ferramenta que permita a tradução automática das especificações descritas no modelo lógico para uma linguagem de programação.

REFERÊNCIAS

[1] CAMOLESI, A.R. *Proposta de um gerador de ambientes para a modelagem de aplicações usando Tecnologia Adaptativa*. Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, São Paulo, 2007.

[2] NETO, J. J. *Contribuições à metodologia de construção de compiladores*. Tese de Livre Docência, USP, São Paulo, 1993.

[3] PISTORI, H. *Tecnologia Adaptativa em Engenharia de Computação:* Estado da Arte e Aplicações. Tese de Doutorado, USP, São Paulo, 2003.

[4] NETO, J.J.; KOMATSU, W. Compilador de Gramáticas Descritas na Notação de Wirth Modificada. *Anais EPUSP* - Engenharia de Eletricidade - série B, vol. 1, pp. 477-517, São Paulo, 1988.

[5] ALMEIDA, J.R. STAD. *Uma ferramenta para representação e simulação de sistemas através de statecharts adaptativos*. Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, São Paulo, 1995.

[6] CAMOLESI, A.R. e NETO, J.J. Modelagem Adaptativa de Aplicções Complexas. XXX *Conferência Latinoamericana de Informática* - CLEI'04. Arequipa - Peru, Setiembre 27 - Octubre 1, 2004a

[7] NETO, J. J. *Adaptive Rule-Driven Devices - General Formulation and Case Study*. Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA 2001, Springer-Verlag, Vol.2494, pp. 234-250, Pretoria, South Africa, July 23-25, 2001.