PML: UMA LINGUAGEM PARA MODELAGEM E AVALIAÇÃO DOS PROCESSADORES

Max Miller Silveira, Márcio Kreutz,

Ana Luisa Medeiros, Jonathan Wanderley

max.silveira@ifrn.edu.br, kreutz@dimap.ufrn.br,

analuisafdm@gmail.com, jonathan.wanderley@gmail.com

Universidade Federal do Rio Grande do Norte/Brazil

Abstract : **Domain Specific Modelling Languages (DSMLs) inherit concepts of model-driven engineering aiming at defining components for software and hardware related to a specific domain. In this context, DSMLs usually are targeted to support more than one level of abstraction, bringing support to better understand the semantics of these components as well as enabling validation about their main concepts and relationships. Regarding hardware components modelling, specific languages may provide support on processing architectures models generation, easing architectural explorations for different configurations which in turn, can be tested against design constraints. In this context, this work presents a Domain Specific Modelling Languages called DSML (Processor Modelling Language) aiming at helping designers to efficiently design processing architectures . The language is supported by a tool for editing and generation of models at different abstraction levels in SystemC. Thus, the generated models can be simulated, allowing functionality and conformity with design constraints exploration.**

Keywords: **EMF; Model-driven engineering; Meta-models; Processors.**

Resumo **– Linguagens para Modelagem de Domínios Específicos fazem uso de preceitos da Engenharia Dirigida a Modelos para a definição de componentes de software e de hardware relacionados a um domínio específico. Nesse contexto, essas linguagens são concebidas para darem suporte a mais de um nível de abstração, além de trazer fácil conhecimento e validação a respeito da semântica e de possíveis relacionamentos entre estes. Em modelagem de componentes de hardware, podem oferecer suporte para geração de modelos simuláveis de arquiteturas de processadores, facilitando o projeto de diversas configurações a serem testadas frente às restrições de projeto. Nesse sentido, esse trabalho apresenta uma Linguagem de Domínio Específico chamada PML (*Processor Modelling Language*), para a especificação de modelos de arquiteturas de processadores, bem como uma ferramenta para edição e geração de descrições de processadores em diferentes níveis de abstração. Essas descrições são simuláveis podendo então serem avaliadas em relação às suas funcionalidades, restrições de projeto e desempenho de simulação.**

Palavras-chave: **DSML; EMF; Engenharia dirigida a modelos; Meta-modelos; Arquitetura de processadores.**

I. INTRODUÇÃO

A Engenharia dirigida por Modelos (Model Driven Engineering, MDE) [1] propõe que descrições de aplicações e arquiteturas ocorram através de linguagens com sintaxe gráfica e em diferentes níveis de abstração. Nessa abordagem, a semântica de componentes de software e de hardware, é expressa através de meta-modelos, os quais especificam o significado das funcionalidades do componente, bem como seus relacionamentos. Dessa forma, meta-modelos podem ser usados para a criação de linguagens que expressem o comportamento de seus componentes, através de modelos, por exemplo, um meta-modelo para processadores, deve conter cada componente que pode pertencer à uma arquitetura e seus relacionamentos. Nesse sentido, uma linguagem que implemente esse meta-modelo irá permitir a descrição de modelos de arquiteturas de processadores.

As Linguagens de Modelagem de Domínio Específico (*Domain-Specific Modelling Languages*, DSMLs) [2] surgem como uma estratégia para utilização dos princípios da Engenharia dirigida por Modelos, no intuito de conceber e descrever componentes de software e de hardware através de uma semântica pertencente a um domínio específico do conhecimento. Adicionalmente, podem prever a representação de componentes em mais de um nível de abstração, bem como permitir uma fácil visualização de relações entre conceitos, entendimento e validação de funcionalidades modeladas.

Existe a possibilidade de criar uma DSML através da criação de uma extensão de uma linguagem já existente, tal como a UML (*Unified Modelling Language*), ou por meio da utilização de ferramentas que manipulem meta-modelos formais. Todavia, alguns pontos sempre devem ser observados, tais como a semântica, as características, as abstrações e as relações entre os elementos que se pretende criar na linguagem [3].

Para que projetistas possam eficientemente conceber e configurar arquiteturas de processadores, os conceitos da engenharia dirigida por modelos podem ser aplicados na criação de uma DSML Nessa perspectiva, este artigo apresenta uma linguagem de modelagem de domínio específico para processadores, chamada PML (Processor Modelling Language), que se propõe a tornar eficiente a concepção e descrição de modelos de processadores em diferentes níveis de abstração. Além disso, pode oferecer suporte a geração dos modelos de processadores através de uma ferramenta para transformação de modelos, que foi desenvolvida. Essa ferramenta é capaz de gerar código em SystemC[4], compatível com os níveis de abstração TLM (*Transaction Level Modelling*) [5].

Este artigo está estruturado da seguinte forma. A Seção 2 descreve conceitos básicos relacionados à Arquitetura de Processadores, o processo de modelagem de componentes de hardware e linguagens de domínio específico. A Seção 3 apresenta a PML e seu meta-modelo. Na seção 4 são apresentados e discutidos resultados experimentais e na seção 5, as conclusões e trabalhos futuros.

II. Conceitos Básicos

A. *O processo de modelagem de componentes de hardware*

A realização da modelagem de processadores em um maior nível de abstração necessita de ciclos de vida mais curtos e uma exploração mais detalhada do projeto [6]. Nesse contexto, o projeto dos componentes de hardware possui as etapas de: (i) Projeto de Modelagem da Arquitetura, (ii) Projeto de Modelagem em nível de Transferência entre Registradores (RTL, Register Transfer Level), (iii) Projeto Físico. Nas etapas (i) e (ii) há a utilização de algum tipo de software, ao contrário da etapa (iii). Dessa forma, na etapa i é realizada a modelagem e simulação de modelos mais abstratos, e na etapa (ii) ocorrem o refinamento, modelagem e simulação para a realização da previsão do desempenho da arquitetura. Após a realização das etapas (i) e (ii), é realizada a especificação lógica, com base na arquitetura que foi modelada. Em seguida, as especificações lógicas são transformadas em circuitos, onde engenheiros de layout posicionam os circuitos na planta baixa do circuito integrado (etapa (iii)) [8].

A modelagem de um sistema de hardware ocorre na forma de um sistema reativo, ou seja, sistemas regidos basicamente por estímulos. Essa modelagem ocorre através de interações com o ambiente, onde o hardware é expresso através de um conjunto de processos que reagem de forma contínua a eventos do ambiente [9]. A reação ou reatividade pode ser representada, em diversas ferramentas que modelam hardware, por sinais e eventos (mudança nos valores do sinal). O modelo de incorporar a reatividade é conhecido como event-driven, que é suficiente para a descrição da maioria do hardware e sistemas, uma vez que, oferece suporte a vários níveis de abstrações [10].

A realização da modelagem abstrata de um processador e/ou qualquer outro componente de hardware permite a obtenção de dados iniciais sobre o desempenho, eficiência e arquitetura do componente. Nesse sentido, na modelagem de um sistema em chip (*System on Chip*, SoC), por exemplo observa-se a necessidade de ambientes de software que viabilizem uma exploração do espaço de projeto de forma rápida, isso pode ser alcançado por meio do provimento de recursos para a modelagem e a simulação dos componentes de um SoC. Dessa forma, com o uso de um SoC é possível a integração de sistemas digitais completos em um único chip, que pode conter a CPU, memória e outros componentes periféricos [11].

Nessa perspectiva, os projetistas utilizam modelos de hardware implementados em softwares de simulação com o intuito de aumentar a velocidade do desenvolvimento de SoC. As ferramentas de simulação permitem simular a execução de programas para a validação da correção e desempenho do que foi projetado. Nesse sentido, é possível que desenvolvedores de software utilizem esses modelos para desenvolvimento e testes de programa.

Todavia, os modelos de simulação são mais lentos que o hardware real, porém eles são construídos e testados de forma rápida, o que resulta na redução do tempo necessário à obtenção de uma arquitetura de hardware otimizada. Dessa forma, no desenvolvimento dos modelos de hardware existem as Linguagens para Descrição de Hardware (*Hardware Description Languages*, HDLs) [12], e as Linguagens para Descrição de Arquiteturas (*Architecture Description Languages*, ADLs) [13], que em conjunto com software de simulação permitem a modelagem e simulação de processadores.

As ADLs servem para especificar arquiteturas, e permitem a realização de simulações para identificar a melhor arquitetura para um conjunto de aplicações. Além disso, podem ser utilizadas na geração de protótipos de hardware, que contenham a especificação das características, tal como a velocidade do relógio. Em outras palavras, as ADLs permitem especificar, simular, avaliar e validar uma arquitetura antes de construí-la. Essas características resultam na economia de tempo e diminuem os custos para a criação de arquiteturas [13].

Já as HDLs são quaisquer linguagens que permitam realizar descrições ou projetos de circuitos eletrônicos, utilizando lógica digital. No processo de descrição é considerado o funcionamento, concepção, organização, e testes para verificar o circuito. Além disso, as HDLs

expressam a estrutura espacial, temporal e comportamental dos sistemas eletrônicos. A sintaxe e semântica das HDLs incluem anotações explícitas para expressar o paralelismo, sendo utilizadas para descrever especificações executáveis de alguma parte de hardware [14].

B. *Linguagem de Modelagem de Domínio Específico*

As Linguagens de Modelagem de Domínio Específico (*Domain-Specific Modelling Languages*, DSML) [15] são utilizadas para a concepção e desenvolvimento de sistemas. Essas linguagens são formadas por sintaxe concreta, sintaxe abstrata, semântica, além de mapeamentos entre sintaxe abstrata e concreta e sintaxe abstrata e semântica.

A sintaxe abstrata é utilizada para a expressão da semântica dos componentes de um domínio específico, bem como dos seus possíveis relacionamentos. Dentro do contexto da Engenharia dirigida a Modelos, geralmente sintaxes abstratas são representadas por meta-modelos. Já a sintaxe concreta serve para permitir a descrição dos componentes e de seus relacionamentos, que podem formar um ou mais modelos.

Na PML a sintaxe abstrata é expressa utilizando-se o metamodelo ECORE [16] o qual por sua vez, pode ser considerado um meta-metamodelo para a UML. Nesse sentido, diagramas de classes UML servem para definir os conceitos da PML, os quais correspondem à componentes de arquitetura de processadores. A Fig. 1 abaixo ilustra alguns desses componentes. Note que as interfaces dos componentes precisam estar bem definidas para se possa compo-los (conectá-los) de modo a formar a estrutura do processador que se está modelando.

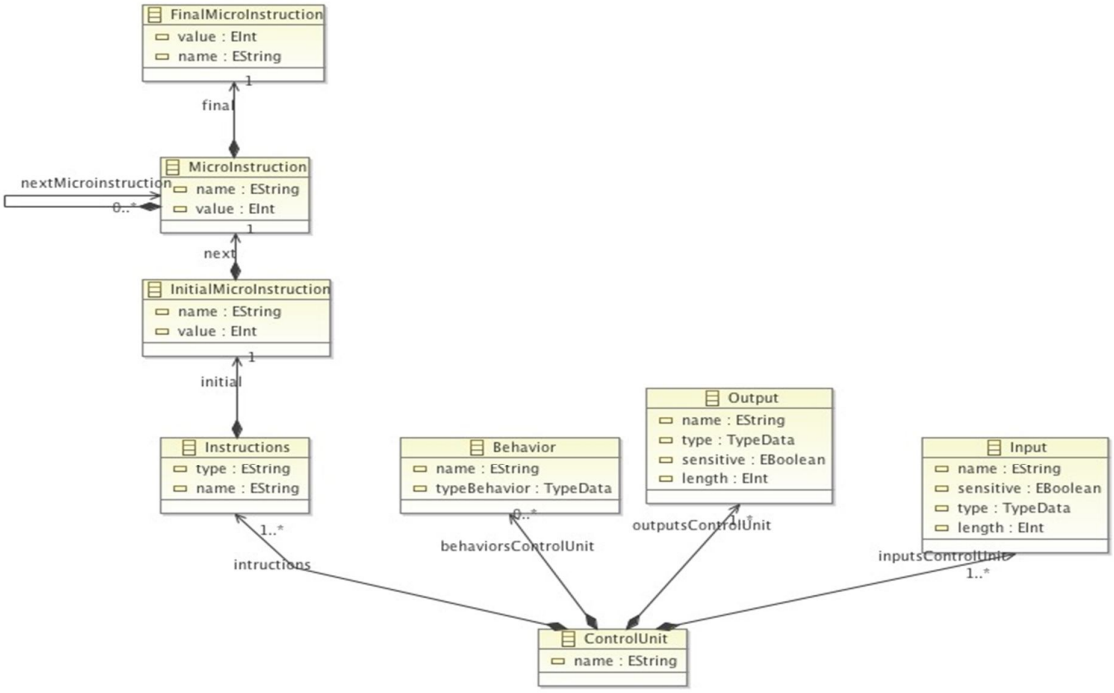

Fig. 1. Modelo da Unidade de Controle descrito na PML

A conexão entre os componentes é realizada através de diagramas de objetos UML, onde uma estrutura completa de processador pode ser definida. A Fig. 2 ilustra a parte operativa de um processador, é importante ressaltar que a figura está simplificada uma vez cada componente conectado ao processador (*Processor*) é formado por uma composição de outros componentes como entradas e saídas.

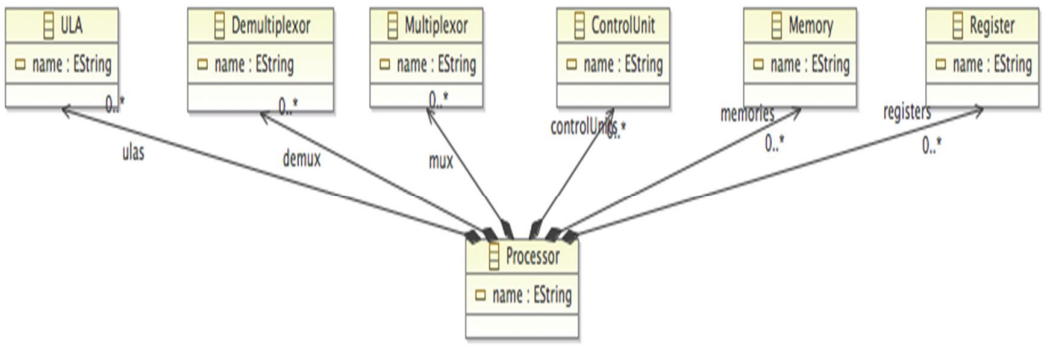

Fig. 2. Modelo simplificado do metamodelo do Processor descrito na PML

A semântica de cada componente é dada pelo estereótipo e pela sua interface. Por exemplo, uma ULA será estereotipada por uma interface "ULA", a qual define suas operações e as portas de entrada e saída, conforme ilustrado na Fig. 3. Além das operações cada estereótipo define também, qual (is) operação (ões) cada evento associado à cada canal de entrada irá executar quando ativo. Essa semântica permite a simulação de circuitos digitais: assim que um valor é modificado em alguma porta de entrada, o circuito ao qual está conectado deve ter suas operações executadas (avaliadas) naquele tempo de simulação.

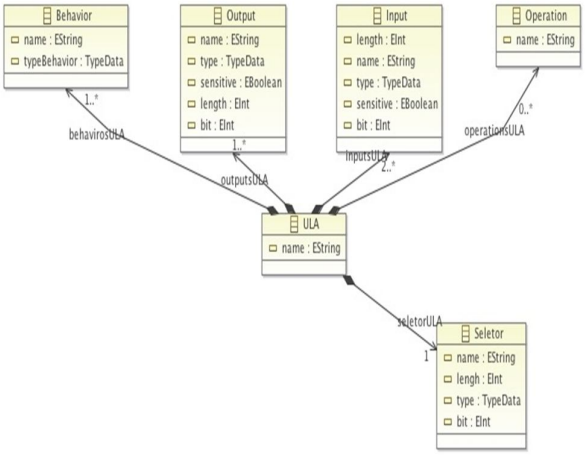

Fig. 3. Metamodelo da ULA na PML

Finalmente, a sintaxe concreta adotada para a PML se refere ao XML definido pelo editor do Eclipse. Nesse editor, cada componente pode ser instanciado e conectado aos demais, conforme exemplo da Fig. 4 que ilustra interface do editor e da Fig. 5 que demonstra o XML gerado pelo editor.

A PML, além de permitir a especificação de arquiteturas de processadores utilizando o Editor Eclipse, gera uma descrição equivalente em SystemC. Como o simulador do SystemC possui a semântica de simulação compatível com o modelo orientado à eventos, a tradução ocorre de forma direta: cada componente e canal do XML é traduzido pelo seu equivalente na sintaxe concreta do SystemC; a semântica permanece mesma. Isso é realizado para que as descrições possam ser simuladas para fins de avaliação funcional e de desempenho.

A transformação entre as sintaxes concretas PML(XML) para SystemC é realizada a através da ferramentas Acceleo em conjunto com um Wrapper Java que serve como uma extensão customizada da ferramenta. Antes de iniciar o processo de transformação se faz necessário descrever o modelo de processador na PML gerando assim um arquivo XML, conforme pode ser observado na Fig. 4 abaixo que é uma representação simplificada de parte de um processador MIPS, sendo nessa descrição composta por uma ULA (tag ulas, linha 03), um banco de registradores (tag registers com atributo length definido, linha 10) e o(s) barramento(s)(tag signalEvent, linha 12).

```
1<?xml version="1.0" encoding="UTF-8"?>
2<pml:Processor              xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:pml="http://pml/0.1" name="newMips">
3  <ulas name="ulaMIPS">
4  <behavirosULA name="execution"/>
5  <outputsULA name="r" type="Int" bit="32"/>
6 <inputsULA name="opA" type="Int" bit="32"/>
7    <operationsULA name="add"/>
8    <operationsULA name="sub"/>
9  </ulas>
10          <registers     name="registerFile"
length="32">
        ...
11  </registers>
12<processorSignalEvents     name="ir_mux_ula"
signalIn="//@mux.1/@inmux"
signalOut="//@registers.1/@outputRegister.1
//@decoders.0/@outDecoder.3"/>
13</pml:Processor>
```

Fig. 4. Descrição de uma ULA na PML

Após a descrição do modelo entra em ação a execução de uma classe Java que foi chamada de Generator (Fig. 5) a qual é gerada pela ferramenta Acceleo. É nesta classe que serão definidas, a localização física do modelo PML(XML) gerado pelo editor do Eclipse, quais templates Acceleo descritos serão usados no processo de tradução do modelo PML para um modelo equivalente em SystemC, o local onde ficará o código gerado pelo Acceleo e qual o template principal da linguagem.

```
1 package PMLGen.generator;
2 import br.ufrn.lasic.pml.PMLPackage;
3public      class      Generator      extends
AbstractAcceleoGenerator {
4 public static final String MODULE_FILE_NAME
="";
5 public static final String P_FILE_NAME ="";
6 public static final String F_NAME ="/pml";

7 public static final String[] TEMPLATE_NAMES=
{};
8 private List<String> propertiesFiles = new
ArrayList<String>();
9 public static void main(String[] args) {
10   try {
11              URI                modelURI=
URI.createFileURI(P_FILE_NAME);
12 File folder = new File(F_NAME);
13          List<String>          arguments=new
ArrayList<String>();
14     Generator     generator     =     new
Generator(modelURI, folder, arguments);
15 for (int i = 2; i < args.length; i++) {
16 generator.addPropertiesFile(args[i]);
17 }
18 generator.doGenerate(new BasicMonitor());
19 } catch (IOException e) {
20            e.printStackTrace();
21        }
22    }
23 }
```

Fig. 5. Classe Generator

Com a execução da classe Generator na linha 16 é iniciado a transformação do modelo PML para SystemC, onde cada elemento descrito na PML passa a ser analisado e traduzido para o equivalente em SystemC. Esse processo de equivalência é realizado no template Acceleo (Figuras 6 e 7) que interpreta cada elemento do XML e o processa conforma as regras descritas no Acceleo.

```
1[comment encoding = UTF-8 /]
2[module ula_h('/PML/model/pml.ecore')]
```

```
3[import PMLGen::generator::pmlutil /]
4[template public generateElement(c : ULA)]
5[file   (c.name.concat('_untimed.h'),   false,
'UTF-8')]
6       #ifndef [c.name.concat('_h')/]
7       #define  [c.name.concat('_h')/]
8       #include "systemc.h"
9       SC_MODULE([c.name/]){
10              typedef enum ULAOPERATION_{
11                 [for   (p   :   Operation   |
c.operationsULA)]
12
        ulaOperation_[p.name/],
13              [/for]
14              }
15[for (p : Input | c.inputsULA)]
16forIn(p.name,p.type.toString(),p.bit,p.lengt
h)/]
17              [/for]
18        [for (p : Output | c.outputsULA)]
19    [forOut(p.name,        :    Behavior    |
c.behavirosULA)]
20
21        [/for]
22   SC_CTOR([c.name/]) {
23 [for (o : Behavior | c.behavirosULA)]
24
        SC_METHOD([o.name.toLower()/]);
25 [/for]
26 [for (p : Input | c.inputsULA)]
27              [if(p.sensitive) ]
28      [forSensitive(p.name,
p.length.abs())/]
29                      [/if]
30                      [/for]
31        }
32      };
33      #endif
34
35      [/file]
36[/template]
```

Fig. 6. Template parcial da ULA em Acceleo

```
1[comment encoding = UTF-8 /]
2[module processor_h('/PML/model/pml.ecore')]
3[import PMLGen::generator::pmlutil /]
4[template    public    generateElement(c    :
Processor)]
5[file    (c.name.concat('_timed.h'),    false,
'UTF-8')]
6       #ifndef [c.name.concat('_h')/]
7       #define  [c.name.concat('_h')/]
8       #include "systemc.h"
9
10      SC_MODULE([c.name/]){
11
12     sc_in_clk clock;
13      [processSignal(c)  /]
14     [for     (p     :     SignalEvent     |
c.processorSignalEvents)]
15       sc_signal< sc_ > > [p.name /];
16
17
18              [/for]
19              [for (p : ULA | c.ulas)]
20              [p.name/]
*[p.name.toLower()/]_;
21              [/for]
22              [for    (p    :    Register    |
c.registers)]
23              [p.name/]
*[p.name.toLower()/]_;
24              [/for]
25          }
26      };
27      #endif
28[/file]
29[/template]
```

Fig. 7. Template parcial do Processador em Acceleo

O template Acceleo neste processo é o responsável por realizar a transformação do modelo

PML para o modelo SystemC. Entretanto, existem algumas limitações na ferramenta Acceleo, como por exemplo, a falta de laços com variáveis: a ferramenta somente permite laços do estilo foreach (laços que percorrem lista de objetos). Contudo, a ferramenta permite que códigos adicionais em Java (Wrappers) possam ser implementados e interligados ao template. Na linha 3 das figuras 6 e 7 pode ser observada a importação de arquivo chamado "pmlutil", o qual contem as descrições das chamadas dos métodos Java implementados no Wrapper conforme a Fig. 8 abaixo.

```
1[comment encoding = MacRoman /]
2[module
pmlutil('http://www.eclipse.org/acceleo/mtl/3.
0',   'http://www.eclipse.org/emf/2002/Ecore',
'/PML/model/pml.ecore')/]
3[query public forIn(arg0 : String, arg1 :
String, arg2 : Integer, arg3 : Integer) :
String
4       =
invoke('PMLGen.wrapper.ForSensitive',
'forIn(java.lang.String,     java.lang.String,
java.lang.Integer,        java.lang.Integer)',
Sequence{arg0, arg1, arg2, arg3}) /]
5[query public processSignal(arg0 : Processor)
: String
6       =
invoke('PMLGen.wrapper.ForSensitive',
'processSignal(br.ufrn.lasic.pml.Processor)',
Sequence{arg0}) /]
```

Fig. 8. Arquivo PMLUTIL

Na linha 16 (forIn) da Fig. 6 e na linha 14 (c.processorSignalEvents) da Fig. 7 ocorre a chamada de métodos implementados no Wrapper Java que é responsável por realizar o processamento de elementos e regras não realizados pelo template Acceleo. O Wrapper Java é ilustrado na Fig. 9.

```
1 package PMLGen.wrapper;
2 import br.ufrn.lasic.pml.*;
3
4 public class ForSensitive {
5 public String forIn(parametros) {
 //Processamento    das    entradas    (Input)

}
6 public String processSignal(Processor p){
 //Processamentos dos signalEvents(Barramento)
7 }
8 }
```

Fig. 9. Wrapper Java

O processo de processamento pelo Wrapper se dá pela chamadas dos métodos da classe ForSensitive (Fig. 9, linha 4) que contém a implementação dos métodos forIn e processSignal. Estes métodos contém a implementação do processo de texto que o Acceleo não é capaz de realizar, retornando para o template o texto processado para ser acoplado na transformação do modelo. O método forIn tem por responsabilidade verificar e gerar, caso necessário, as entradas sensíveis à mudanças de estado: por exemplo, a partir da linha 16 da Fig. 6 é gerado o código equivalente à linha 17 da Fig. 10. Atém disso, o método processSignal tem como uma das funções realizar a conexão entre os elementos, então o código da linha 14 da Fig. 7 irá resultar no código da linha 18 e 24 da Fig. 11 que é a conexão entre os elementos ao barramento.

Ao final do processo de execução dos templates será gerado um código SystemC (Fig. 10 e 11) equivalente ao que foi descrito na linguagem PML, de modo a ser possível simular e executar o código para analisar o desempenho do processador gerado.

```
1       #ifndef ULAMips_h
2       #define  ULAMips_h
3       #include "systemc.h"
4       SC_MODULE(ULAMips){
5               typedef enum ULAOPERATION_{
6               ulaOperation_add,
7               ulaOperation_sub,
8               }
9                  sc_in<           sc_int<32>
>inputULAMips;
10                 sc_in<           sc_int<32>
>inputULAMipsB;
11                sc_out<           sc_int<32>
>resultULAMips;
12                 sc_out<zeroULAMips;

13              sc_in<              sc_int<5>
>operationULAMips;
14          void execute();
15      SC_CTOR(ULAMips) {
16                      SC_THREAD(execute);
17              sensitive <<inputULAMips;
18               sensitive <<inputULAMipsB;
19      }
20      };
21      #endif
```

Fig. 10. Código da ULA gerado em SystemC

```
1       #ifndef newMips_h
2       #define  newMips_h
3       #include "systemc.h"
4
5       SC_MODULE(newMips){
6              AddIR *addir_;
7              ULAMips *ulamips_;
8              muxDecode *muxdecode_;
9              muxIR *muxir_;
10             muxULAMips *muxulamips_;
11           InstructionMemory
*instructionmemory_;
12             DataMem *datamem_;
13             PC *pc_;
14             registerFile *registerfile_;
15              sc_in_clk clock;
16              sc_signal< type >ir_ula;
17              ULAMips->inputULAMips(ir_ula);
18              sc_signal< type >mux_ula;
19              ULAMips-
>inputULAMips(mux_ula);
20              sc_signal< type >dataMemUla;
21              ULAMips-
>resultULAMips(dataMemUla);
22              sc_signal< type >irdatamem;
23              ULAMips-
>resultULAMips(mux_ula);
24                SC_CTOR(newMips) {
25                 }
26      };
27      #endif
```

Fig. 11. Código do processador contendo a ULA em SystemC

III. LINGUAGEM PARA MODELAGEM DE PROCESSADORES

A Linguagem de Modelagem de Processadores (PML) é uma DSML para processadores, que define a sintaxe abstrata, concreta e a semântica

para o domínio de processadores. Além disso, essa linguagem é capaz de gerar modelos compatíveis com ADLs e HDLs. Nesse sentido, a Fig. 4 representa um modelo do processador MIPS que a PML pode ser utilizada para descrever.

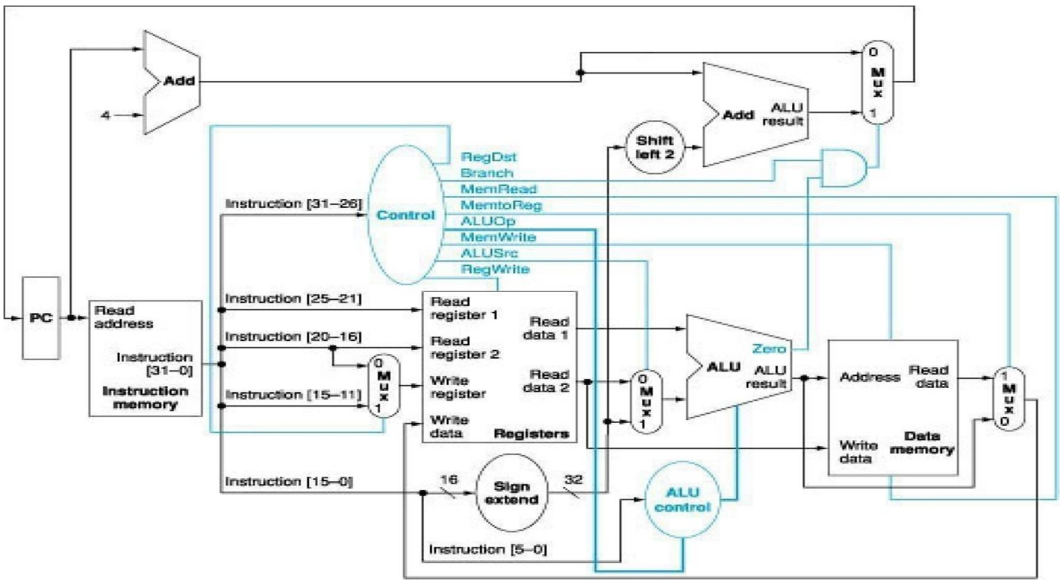

Fig. 12. Arquitetura do Processador MIPS[23].

A Fig. 12 acima que representa a arquitetura de um processador MIPS qualquer pode ser modelada na PML através dos componentes que foram apresentados na Fig. 2 da seção II.B. Por exemplo a unidade lógica e aritmética deste deste processador para ser modelada utiliza-se da representação da ULA descrita na linguagem PML conforme ilustra a Fig. 11 que representa a parte operativa da ULA e a Fig. 12 que representa o barramento dos componentes. O modelo da ULA foi ilustrado na Fig. 4 onde as linhas 03 a 33 representa os componentes ULA e registradores e as linhas 34 a 41 representa o barramento que a ULA e os registradores estão conectados.

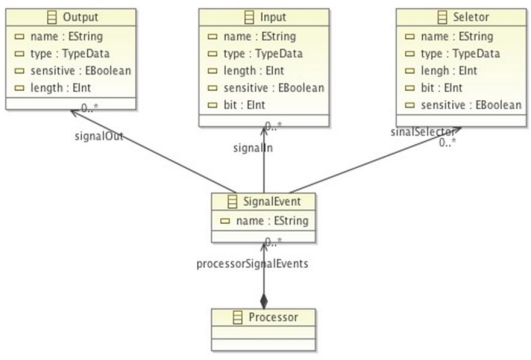

Fig. 13. Metamodelo do Barramento de conexão na PML

A sintaxe da PML é baseada nos conceitos apresentados na seção II.A, que define os conceitos relacionados à processadores. Nesse sentido, os processadores são compostos basicamente pelos seguintes componentes:

- Multiplexador
- Demultiplexador
- Decodificador
- Registrador
- Unidade Lógica e Aritmética
- Memória
- Unidade de Controle

A sintaxe abstrata e concreta da PML foi definida através da meta-linguagem EMF (Eclipse Modeling Framework) [16], onde os elementos da sintaxe concreta são a representação de cada componente, tanto na forma gráfica como textual. A sintaxe abstrata são as restrições, relacionamentos e validações realizadas entre os componentes.

No contexto de uma ULA, as descrições das entidades pertencentes à PML Input (Entrada de um componente), Output (Saída de um componente) e Seletor (Seleciona um entrada ou saída de acordo com o componente), possuem os seguintes atributos:

- Name: representa o nome da entidade;
- Type e TypeBehavior : representa o tipo de dado relacionado à entidade;
- Sensitive: elemento utilizado para representar se uma entidade pode ser sensível a um evento;
- Length: representa o tamanho da entidade em quantidade, por exemplo, uma entidade que representa uma entrada de dados, a propriedade

length sendo preenchida com tamanho cinco, significa que terá cinco entradas com aquelas características;

- Bits: representa a quantidade de bits que o componente pode conter.

Ainda neste contexto a entidade Operation que representa as operações que a ULA irá executar, é importante ressaltar que no caso das operações da ULA após o código gerado é necessário colocar a lógica de programação de cada operação. Já a entidade Behavior representa os métodos que os componentes executam de acordo com os eventos que ocorrem. Além disso, possui os atributos Name e Type.

Na PML a semântica é definida através dos modelos de computação, que definem a semântica para o domínio de processadores, mais especificamente os modelos de eventos e os modelos de máquina de estados finitos. No modelo de eventos a semântica foi representada através dos seguintes elementos:

Propriedade sensitive nas entradas de cada componente (representado pelas entidades Input e Seletor), o que indica que ao ocorrer um evento em sua entrada poderá iniciar o processamento do dado no componente, ativando as execuções descritas nos Behavior do componente.

Interligação dos componentes através da entidade SignalEvent, que é a responsável por representar a interligação entre os componentes através de um canal de comunicação qualquer (ver Fig. 13).

O padrão TLM 2.0 não contempla o modelo com precisão de ciclo (Cycle Accurate), apesar de possuir algumas ferramentas que possibilitem o uso desta versão OSCI 2009[5]. Outro aspecto a se considerar é que no modelo de tempo aproximado (Approximate Time) os canais de comunicação são mais abstratos, consequentemente disparando um número maior de operações à cada evento. Os modelos Cycle Accurate, por sua vez, executam todas as operações a cada ciclo de relógio, tornando a simulação potencialmente mais precisa e com um tempo de execução maior.

Os modelos simuláveis gerados nas arquiteturas modeladas usando a PML são compatíveis com os níveis TLM de tempo aproximado (Approximate Time) e precisão de ciclo (Cycle Accurate):

- Modelo compatível com o Approximate Time: a compatibilidade ocorre através do uso de portas e interfaces de comunicação, com uma contagem de tempo relativa e aproximada;

- Modelo compatível com o Cycle Accurate: modelo bem próximo do nível RTL, com uso de ciclos de clock como elemento que gerencia os eventos que ocorrem, a comunicação é feita utilizando sinais, declarados como "sc_signal".

As diferenças entre os dois modelos gerados são descritas com base na granularidade dos canais de comunicação e da passagem do tempo. A comunicação no modelo Approximate Time é realizada através da implementação de canais de comunicação abstratos, os quais definem eventos com a semântica estabelecendo a execução de um conjunto de operações. Esse conjunto de operações pode representar a execução de vários ciclos de relógio. Já no modelo Cycle Accurate a comunicação é realizada através de sinais (sc_signal), que representam fios, cujos eventos internos definem operações que ocorrem à cada ciclo de relógio. A Fig. 14 representa a diferença entre os modelos.

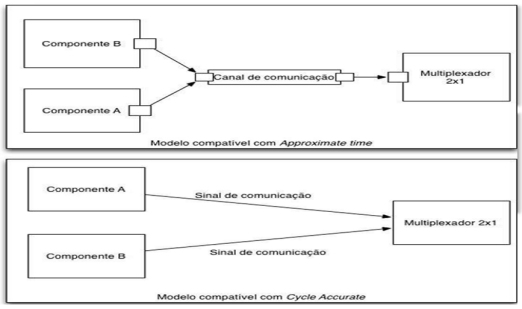

Fig. 14. Diferença entre os modelos.

De acordo com a Fig. 13, é possível observar que no modelo *Cycle Accurate* as portas de comunicação entre os componentes são fios (sinais), enquanto que no modelo *Approximate Time*, canais abstratos.

```
01SC_MODULE(Component_AT){//Approximate Time
02   sc_in <bus_channel<int>> inport; entrada
03
04   int x;
05
06   void execution_method_TLM(){
07      x = inport.read();
08      // operations at method level
09    }
10
11    Component_AT() { // constructor method
12      SC_METHOD(execution_method_TLM);
13      sensitive<<
bus_channel.get_default_event();
14    }
15 }
16 SC_MODULE(Component_CA){//Cycle Accurate
17
18  sc_in_clk clk;
19 sc_in <sc_uint<32>>inport;//porta 32 bits
20
21   int x[32];
22
23   void execution_method_CA(){
24      for(int r=0; r < 32; r++) {
25         if(clk == 1) {
26             x[r] = inport.read(r);
27            // operations at cycle level
28         }
29      }
30    }
31
32    Component_CA() { // constructor method
33      SC_METHOD(execution_method_CA);
34      sensitive << clk;
35    }
36 }
```

Fig. 15. Pseudocódigo Approximate Time e Cycle Accurate

A Fig. 15 mostra exemplo de pseudocódigos para os modelos *Approximate Time* e *Cycle Accurate*. As diferenças entre os níveis de abstração são mostrados nos códigos: linhas 2 e 7, para o modelo *Approximate Time*, e linha 18, 19 e 21 para o modelo *Cycle Accurate*, correspondentes aos componentes “Component_AT” (nível de abstração TLM) e “Component_CA” (nível de abstração *Cycle Accurate*). Nesse sentido, o que diferencia as descrições nesses dois níveis de abstração é a granularidade das operações e dos canais de comunicação.

Quanto às operações, pode-se verificar nas linhas 18 e 25 da Fig. 15, que no nível *Cycle Accurate* as operações são executadas à cada novo pulso de relógio. Dessa forma, se uma operação necessita de 50 ciclos de relógio para executar, todos esses 50 pulsos serão simulados; na linha 27 essa operação é chamada de operação em nível de ciclo (*cycle level*). No nível TLM por sua vez, cada operação - independente de numero de ciclos que necessita para executar - pode ser executada apenas uma vez, através da lógica do método que a implementa; na linha 8, essa operação é chamada de operação em nível de método (*method level*).

4)

No que se refere aos canais de comunicação, a granularidade se manifesta na semântica dos eventos atrelados à estes. Um canal TLM estabelece eventos que disparam a execução de métodos de granularidade variada. Por exemplo, na linha 2 uma porta de entrada é conectada à um canal de comunicação chamado "*bus_channel*", cujo evento dispara a operação do método "*execution_method_TLM*" à cada vez que o evento padrão ("bus_channel.get_default_event()") do canal "*bus_channel*" é ativado (linhas 12 e 13). Já no nível *Cycle Accurate*, a execução do método “*execution_method_CA*” ocorre à cada ciclo de relógio (linhas 33 e 34).

IV. RESULTADOS

A funcionalidade da PML foi avaliada através da modelagem de três processadores, e geração de código SystemC simulável. Os três processadores modelados foram:

1) MIPS sem pipeline (ver Fig. 12; seção III);
2) MIPS com pipeline (Fig. 16); e
3) ARM DLX (Fig. 17).

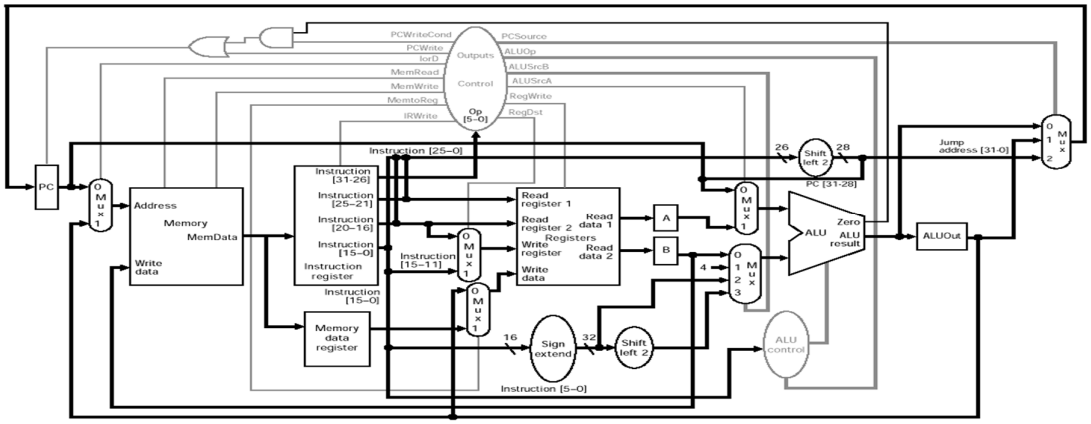

Fig. 16. Processador MIPS Com Pipeline

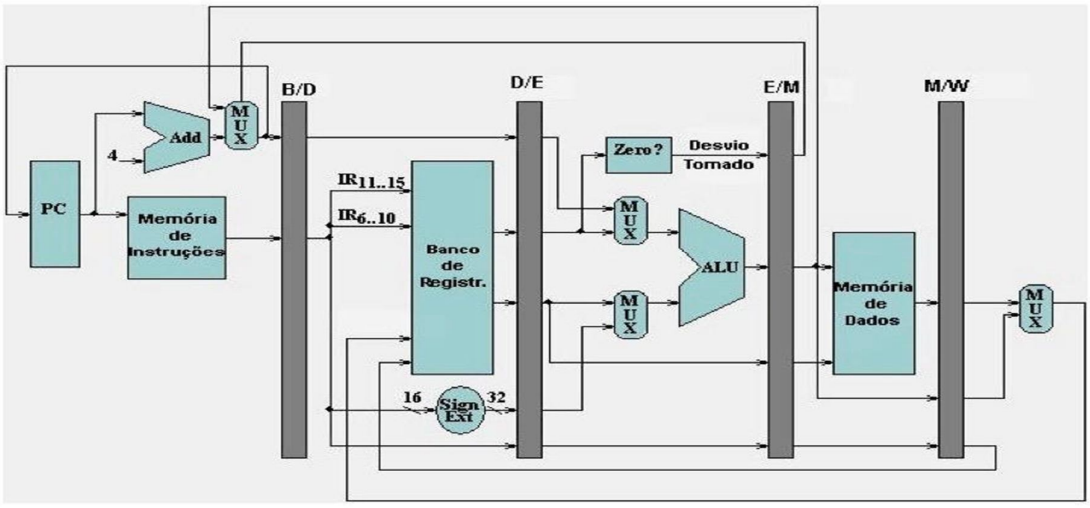

Fig. 17. Processador ARM DLX

No que se refere às descrições em precisão aproximada (*Approximate Time*), foram realizadas duas implementações: (i) todos os componentes internos da micro-arquitetura do processador foram modeladas como um módulo SystemC (*SC_MODULE*) exclusivo, e; (ii) alguns componentes como multiplexadores, por exemplo, foram modelados por lógica de programação (incorporados dentro de outros módulos). Esse segundo tipo de modelo resulta em descrições um pouco mais abstratas, uma vez que os eventos desses componentes não precisam ser gerenciados e escalonados pelo mecanismo de simulação. Os modelos foram chamados de M1, M2, e M3, e representam:

- M1: representa o modelo *Approximate Time*, abstraindo alguns componentes e utilizando canais de comunicação abstratos;
- M2: representa o modelo *Approximate Time*, implementando todos os componentes e canais de comunicação abstratos;
- M3: o modelo *Cycle Accurate*.

Adicionalmente, foi realizada a implementação da lógica referente ao algoritmo para a execução das operações internas dos componentes; para a geração do modelo completo de forma automática, seria necessária também, a geração da Máquina de Estados Finitos da Parte de Controle do processador, bem como das micro-instruções para cada instrução. Nesse sentido, a geração de código-fonte neste trabalho é focada na estrutura da arquitetura do processador, seus componentes e interconexões.

Para a obtenção dos resultados, simulações da execução de instruções foram realizadas nos modelos. As instruções dos programas simulados foram manualmente colocadas em uma descrição de memória ROM, uma vez que até o presente momento não foram utilizados *cross-compilers* para estes modelos. Para fins de teste e para verificar que os modelos de processadores gerados são completamente funcionais, foram simulados todos os tipos de instrução: de controle (laços), de acesso à memória e lógico/aritméticas. Foi simulado um laço (*loop*), onde à cada passada é realizada um acesso à memória e a execução de uma operação aritmética sobre registradores. Ao final do laço, é realizada uma escrita na memória.

A Fig. 18 mostra os resultados sobre o tempo de simulação para os 3 processadores, enquanto que a Fig. 19 mostra os ciclos de relógio (tempos de simulação) necessários para cada processador executar as instruções simuladas.

Quanto ao tempo de simulação, verifica-se claramente a vantagem que pode ser obtida quando modelos mais abstratos são simulados. Percentualmente, modelos M1 simulam em média 300% mais rapidamente do que M3. Esses tempos não são significativos em termos absolutos, porém é verificado que a proporção será mantida em execuções com número maior de instruções, o que comprova a eficiência do nível de abstração empregado na simulação das descrições. Em sistemas complexos, onde um grande número de instruções são executadas, pode-se obter significativa redução do tempo necessário para a exploração de diferentes modelos de arquiteturas.

Quanto ao número de ciclos necessários à execução das instruções, percebe-se que os mesmos estão de acordo com o nível de abstração utilizado. Nos modelos M3 é realizada a simulação com precisão de ciclo, ou seja, ali estão presentes todos os ciclos de relógio necessários à execução de cada micro-instrução. Já nos modelos M1 e M2, cada "ciclo" na verdade, corresponde à 1 tempo de simulação, o qual por sua vez, executa operações que podem ter mais de um ciclo real de execução. Por isso, menos tempos de simulação são necessários para completar a execução das

instruções. Finamente, percebe-se que a descrição do processador MIPS com *pipeline* consome mais tempo do que a versão sem *pipeline*. Isso se deve aos tempos para acesso à memória e para a execução das instruções de controle que provocam a inserção de bolhas no pipeline.

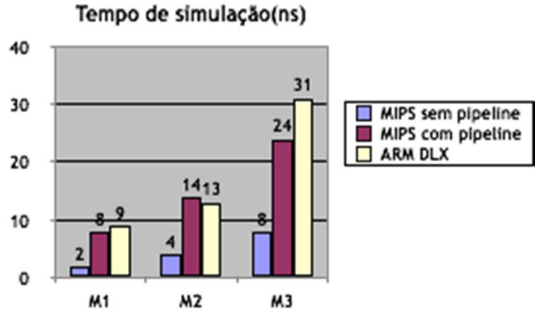

Fig. 18. Resultados do tempo de simulação.

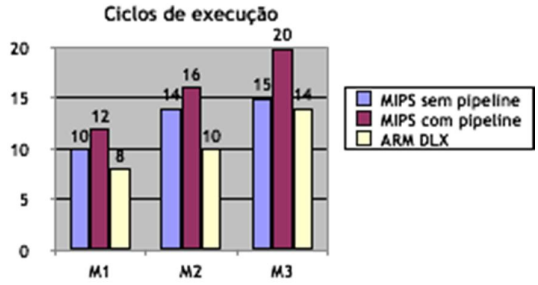

Fig. 19. Resultados da quantidade de ciclos de execução.

As implementações realizadas revelam uma característica interessante da PML, que é possibilidade de reutilização da descrição de componentes para a geração de diferentes modelos de um mesmo processador. Essa possibilidade é altamente desejável no projeto de arquiteturas, uma vez que, permite a exploração de diferentes processadores em um tempo reduzido. Isso ocorre por que processadores compartilham elementos em comum, como por exemplo, multiplexadores, Unidade Lógico-Aritmética, etc. Dessa forma, o uso da PML pode ser um facilitador, no que diz respeito à descrição e geração de modelos de processadores. A Tabela I compara as características da PML com outras linguagens/ferramentas que também permitem a modelagem de processadores.

TABELA I. - COMPARAÇÃO DA PML COM OUTRAS LINGUAGENS/FERRAMENTAS

	DSML	**Modela Processadores**	**Modela outros tipos de hardware**	**Modela o pipeline**	**Geração de código**	**Transformação para outros modelos**
PML	Sim	Sim	Não	Parcial	Sim	Sim
SystemC	Não	Sim	Sim	Sim	Não	Não
ArchC	Não	Sim	Sim	Sim	Não	Não
VHDL	Não	Sim	Sim	Sim	Não	Não
LISA	Não	Sim	Sim	Sim	Não	Não
nML	Não	Sim	Sim	Sim	Não	Não

De acordo com a Tabela I, a PML possui a vantagem de permitir a transformação entre modelos. Além disso, pode ocorrer a criação de novos templates de transformação de modelos, os quais podem ser usados na definição de modelos para outras linguagens tais como VHDL, Verilog, entre outras, ou que se enquadrem na sintaxe e semântica da PML.

Nesse sentido, a PML permite criar não somente modelos em outras linguagens, mas modelos em diferentes níveis de abstração, uma vez que é o template de transformação que determina o grau de abstração do modelo transformado. Dessa forma, a PML é capaz de gerar não somente modelos compatíveis com TLM, bem como modelos em níveis com precisão de ciclo.

Além disso, a PML permite a geração de modelos em ADL e HDL, bastando ser definido o template adequado. Com o objetivo de expor o processo de geração dos modelos e a importância dos templates no processo, a Fig. 21 ilustra a relação dos templates com todos os elementos da linguagem PML, gerados por meio do uso da ferramenta proposta neste trabalho.

Nesse contexto, a Fig. 20 mostra a relação de todos os componentes que compõem o processo de geração de modelos de processadores a partir da PML, onde a linguagem oferece suporte a descrição de vários modelos de processadores. Além disso, através do *Acceleo* juntamente com os *Wrappers* Java, são gerados modelos representados em código, por exemplo, SystemC. É importante ressaltar que são os templates *Acceleo* que são os responsáveis por definir a linguagem em que o modelo será gerado.

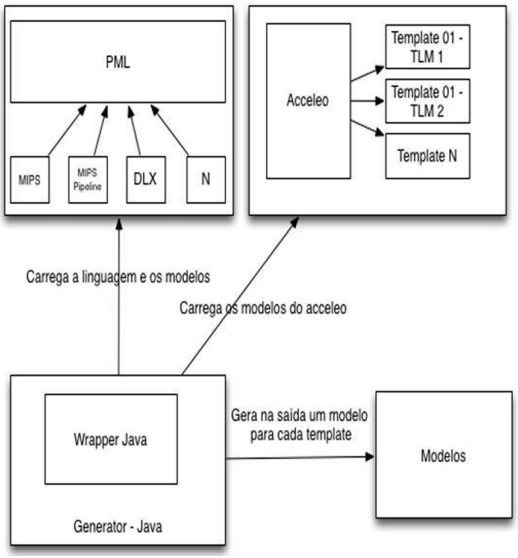

Fig. 20. Processo de geração da PML - inclusão de novos templates.

Os resultados apresentados mostram que a utilização da PML facilita o processo de modelagem de processadores e redução do tempo de modelagem. Adicionalmente, torna possível a reutilização de parte dos códigos de descrição, de forma que alterações nos modelos sejam facilmente realizadas. Além disso, a PML se encontra numa etapa onde o foco é a modelagem estrutural da arquitetura: componentes e suas interconexões. Nesse contexto, devido a PML se encontrar na etapa de concepção da arquitetura, é possível identificar eficazmente arquiteturas e modelos de processadores adequados para cada classe de aplicações alvo.

V. CONCLUSÕES E TRABALHOS FUTUROS

A PML é uma linguagem de modelagem que facilita o processo modelagem e simulação de processadores. Além disso, permite o reuso de parte das modelagens, o que resulta na diminuição no tempo para descrição, modelagem e simulação de processadores.

Adicionalmente, a PML pode permitir que a partir de um modelo descrito na linguagem, ocorra a geração para outros modelos simuláveis. Além disso, a PML permite a descrição de processadores com uma grande quantidade de elementos, porém, por usar uma sintaxe textual, encontra limitações para a visualização da arquitetura, principalmente da estrutura dos componentes arquiteturais modelados e suas interconexões. Como trabalho futuro, essa limitação será contornada com a integração dos elementos de modelagem da PML com linguagem baseada sintaxe gráfica, como UML.

Por fim, este artigo trouxe muitas informações relacionadas ao projeto de hardware baseado nos preceitos da Engenharia Baseada em Modelos, bem como a forma que estes podem ser utilizados para a modelagem de um domínio específico. Características desejadas em projeto de processadores, como geração automática de código e descrições em diferentes níveis de abstração também foram discutidas.

De maneira geral, verifica-se que existem diversas formas para modelagem de componentes de hardware através de meta-modelos ou de linguagens definidas para este propósito. Em relação às linguagens para descrição de hardware, as mais utilizadas são VHDL e Verilog. No entanto, quando o objeto a ser modelado possui natureza complexa, como processadores, o uso dessas linguagens requer que diversos detalhes da arquitetura sejam considerados na descrição, resultando em um tempo maior de projeto e possibilidade de aumento de erros. A abordagem do SystemC permite que, através do uso de uma API C++, arquiteturas sejam modeladas em níveis de abstração mais altos, através do padrão TLM.

A PML permite criar modelos de processadores em um nível de abstração compatível com o TLM. Além disso, foi averiguado que a PML é viável para descrever processadores, bem como utilizar transformação de modelos para SystemC em diferentes níveis de abstração. Essas características possibilitam a criação de diferentes modelos e tipos de processadores, compatíveis com o padrão TLM.

Como trabalhos futuros pode-se citar a expansão da PML para permitir a criação de redes de processadores, gerando sistemas multiprocessados completos, os quais incorporam um conjunto de processadores, arquiteturas de memórias e de comunicação. Além disso, será realizada a integração da PML com o perfil UML-MARTE (*Modeling and Analysis of Realtime Embedded Systems*)[18], visando compatibilizar a linguagem com diversas ferramentas para projeto de sistemas embarcados e de tempo-real.

REFERENCES

1. D.C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," Computer , vol.39, no.2, Feb. 2006, pp.25,31. Doi: 10.1109/MC.2006.58
2. E. Jackson and J. Sztipanovits, Formalizing the structural semantics of domain-specific modeling languages. Software & Systems Modeling, 8(4), 2009, pp. 451-478.. Doi: 10.1007/s10270-008-0105-0
3. K. Chen , J. Sztipanovits and S. Neema. "Toward a semantic anchoring infrastructure for domain-specific modeling languages." Proceedings of the 5th ACM international conference on Embedded software. ACM, 2005.
4. S. Liao, G. Martin, S. Swan and T. Grötker, System Design with SystemC™. Ed.. Springer Science & Business Media, 2002.
5. J. Aynsley, Osci tlm-2.0 language reference manual. Open SystemC Initiative, v. 24, 2009.
6. A. S. Tanenbaum and W. L. Zucchi, Organização estruturada de computadores. Pearson Prentice Hall, 2009.
7. W. Stallings, Arquitetura e organização de computadores: projeto para o desempenho. Prentice-Hall, 2009.
8. S. S. Mukherjee, et al. "Performance simulation tools." Computer. 2002, pp. 38-39.
9. G. Berry, "Real time programming: Special purpose or general purpose languages." IFIP, 1989, San Francisco.
10. S. Liao, S. Tjiang, and R. Gupta. "An efficient implementation of reactivity for modeling hardware in the Scenic design environment. Proceedings of the 34th annual Design Automation Conference. ACM, 1997.
11. T. S. Kumar, R. Govindarajan, and C. P. Ravikumar, On-chip memory architecture exploration framework for DSP processor-based embedded system on chip. ACM Transactions on Embedded Computing Systems (TECS), 11(1), 5, 2012.
12. P. Viana, et al. "Exploring memory hierarchy with ArchC." Computer Architecture and High Performance Computing, 2003. Proceedings. 15th Symposium on. IEEE, 2003.
13. P. Mishra, A. Shrivastava, and N. Dutt. "Architecture description language (ADL)-driven software toolkit generation for architectural exploration of programmable SOCs." ACM Transactions on Design Automation of Electronic Systems (TODAES). Vol. 11. No. 3. ACM, 2004.
14. K. J. Lieberherr, "Towards a standard hardware description language." Proceedings of the 21st Design Automation Conference. IEEE Press, 1984.
15. V. A. Deursen, P. Klint, and J. Visser, Domain-Specific Languages: An Annotated Bibliography. Sigplan Notices, 35(6), 2010, pp. 26-36.
16. D. Steinberg, et al. EMF: eclipse modeling framework. Pearson Education, 2008.
17. V. P. Rubio and J. Cook, A FPGA Implementation of a MIPS RISC Processor for Computer Architecture Education. Diss. MSc. Thesis, New Mexico State University, Las Cruses, New Mexico, America, 2004.
18. M. Faugère, et al, "Marte: Also an uml profile for modeling aadl applications." Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on. IEEE, 2007