MODELAGEM E DESCRIÇÃO DE SISTEMAS COMPUTACIONAIS - UM ESTUDO DE CASO DE COMPARAÇÃO DAS LINGUAGENS VHDL E SDL

César A. M. Marcon, Ney L. V. Calazans, Fernando G. Moraes, Luís H. L. Ries, Fabiano Hessel
{marcon, calazans, moraes, ries, hessel}@inf.pucrs.br

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) - Faculdade de Informática (FACIN)
Av. Ipiranga, 6681 - Prédio 30 / Bloco 4 - CEP 90619-900 - Porto Alegre - RS - BRASIL
Fone: +55 51 3320-3611 - Fax: +55 51 3320-3621

ABSTRACT

This paper discusses several aspects of the modeling and description of computational systems. A metamodel for representing the process of obtaining an abstract description (i. e. a model) of a computational system is proposed and justified. To concretely illustrate the compromises involved in modeling computational systems, the paper shows a case study of specifying a telecom hardware module and describing it in two languages with widely different modeling assumptions, namely VHDL and SDL. The comparison of the resulting descriptions shows some of the strengths and weaknesses of each language in the chosen context. Thus, the adequate choice of languages to use for specifying and describing complex computational systems is showed to be a critical step of the implementation of technological products.

1 INTRODUÇÃO

A crescente complexidade de produtos eletrônicos tecnológicos conduz à elevação cada vez maior do nível de abstração de atividades como especificação, descrição e validação dos sistemas computacionais que os compõe. O objetivo primordial é reduzir, ou pelo menos tornar gerenciável, a complexidade do processo de projetar tais produtos. O principal efeito desejado é o encurtamento do tempo necessário para a chegada de produtos ao mercado (em inglês, ”time to market”), este sendo o principal fator determinante na lucratividade de um produto tecnológico [21].

Sistema computacional é definido aqui como um sistema digital composto por componentes de *hardware* e *software*. Para elevar o nível de abstração destes, utiliza-se ferramentas de apoio ao projeto que aceitam especificações em linguagens como SystemC [11], SpecC [7], Esterel [3], VHDL [20], SDL [16] e Java [17], entre outras. Não há consenso quanto a qual seja a melhor linguagem para a especificação de sistemas computacionais em geral. Faz sentido supor que as diferentes linguagens sejam adequadas aos domínios de aplicação originais para os quais foram concebidas, nenhuma delas sendo capaz de superar todas as demais em uma gama ampla de aplicações. Logo, a comparação de linguagens é uma atividade que pode trazer subsídios para apoiar a escolha de sistemas de projeto e/ou ferramentas de especificação em áreas de aplicação específicas.

A análise comparativa de linguagens de especificação requer um conhecimento aprofundado de modelos computacionais, bem como do processo de elaboração de um modelo de sistema computacional a partir de modelos computacionais genéricos. Além de lidar com os aspectos de modelagem durante a comparação de linguagens de especificação, é necessário validar a comparação das linguagens pela implementação de estudos de caso de sistemas computacionais reais. Tal atividade fornece subsídios práticos para indicar quão bem os modelos subjacentes às linguagens de especificação são capturados e expressos pelas estruturas sintáticas e pela semântica destas linguagens.

Este trabalho propõe um metamodelo para descrever o processo de construção de modelos computacionais e seu emprego na modelagem de sistemas computacionais reais. Adicionalmente, propõe um estudo de caso de descrição de um sistema computacional real em duas linguagens, VHDL e SDL, comparando a adequabilidade das mesmas para o tipo de aplicação em vista, qual seja, a descrição de sistemas de telecomunicação.

A escolha de VHDL como linguagem de especificação do estudo de caso parece facilmente justificável. Trata-se de um sistema freqüentemente implementado em hardware e VHDL se destaca pela sua ampla utilização neste contexto. O uso de SDL, por outro lado, merece uma justificativa. SDL é uma linguagem originalmente elaborada para a especificação de sistemas de telecomunicação e muito usada nesta área de aplicações [5].

A estrutura do restante deste artigo é a seguinte. A Seção 2 propõe e justifica o metamodelo. A Seção 3

introduz as linguagens SDL e VHDL e relaciona estas com modelos computacionais subjacentes. A Seção 4 apresenta o estudo de caso e a comparação entre as descrições. Finalmente, na Seção 5 são apresentadas conclusões e perspectivas para trabalhos futuros.

2 MODELAGEM DE SISTEMAS COMPUTACIONAIS

Abstração é a seleção de características e propriedades essenciais e a correspondente exclusão de outras características que não são relevantes em um dado contexto. Um *modelo* é uma abstração de um sistema real, empregado para viabilizar o raciocínio sobre este sistema. Um *modelo computacional* (em inglês, *model of computation* ou MOC [22]) é um modelo de um sistema composto por hardware e software. Os modelos computacionais servem para formalizar as características de uma classe de sistemas, habilitando a manipulação computacional dos elementos desta classe. Uma *descrição* de um sistema computacional é uma representação gráfica do seu modelo. Note-se a distinção importante entre os conceitos de modelo e descrição. Enquanto descrição designa representações concretas, um modelo é uma representação abstrata. Por exemplo, dado o modelo de computação máquina de estados finita síncrona, existem diversas descrições possíveis de instâncias de sistemas modelados por estas, tais como diagramas de transição de estados síncronos, tabelas de transição de estados síncronas, ou um texto em VHDL.

2.1 Um Metamodelo para o Processo de Modelagem

A partir de uma especificação informal de um sistema computacional, projetistas extraem as características mais relevantes do sistema e produzem uma primeira especificação formal do sistema. Este processo pode ser dividido para fins de estudo nas fases de modelagem (obtenção do modelo do sistema) e de especificação formal (obtenção da descrição inicial, concreta do sistema. Ambas fases são críticas para a qualidade final do sistema computacional em implementação, e são sempre concomitantes.

A Figura 1 ilustra a proposta de *metamodelo* para representar a fase de modelagem de sistemas computacionais. Este metamodelo congrega os conceitos fundamentais de *requisitos de modelagem*, *primitivas de modelagem* e *modelos computacionais*. Estes conceitos, bem como o relacionamento entre eles, mostrado na Figura, define o processo de construção de modelos de sistemas computacionais. O metamodelo pressupõe que todo modelo de um sistema computacional parte da extração de *requisitos de modelagem* que caracterizam o sistema computacional sob implementação, e que são atendidos por *primitivas de modelagem*. Estas primitivas, por sua vez, são usadas na composição de *modelos computacionais* genéricos (MOCs). Finalmente, os modelos computacionais, isoladamente ou em conjunto com outros MOCs, habilitam a representação do *modelo do sistema computacional* almejado.

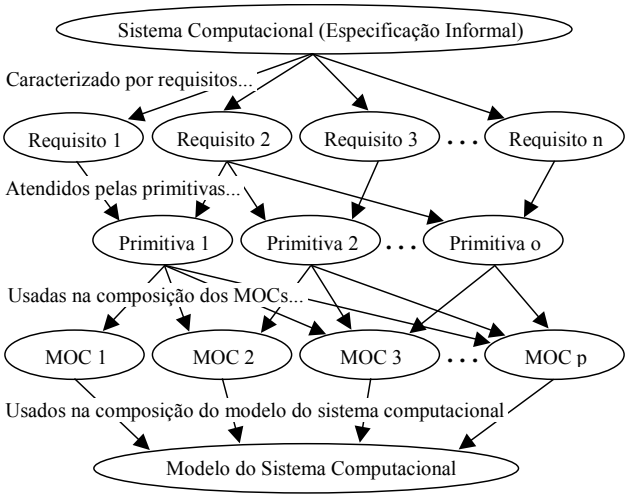

Figura 1 - Metamodelo para a construção de modelos de sistemas computacionais.

As definições relevantes ao metamodelo proposto são:

- ***Requisito de modelagem*** – é uma necessidade, seja de expressar características intrínsecas (e. g. restrições de freqüência de operação, atraso de computação, potência dissipada), seja de descrever uma classe de estruturas ou comportamentos encontrados com freqüência suficiente nos sistemas ou nas classes de sistemas a serem construídos (e.g. operação síncrona ou assíncrona, operação paralela ou seqüencial, comportamento reativo ou interativo);
- ***Primitiva de modelagem*** – é um elemento fundamental que, isoladamente, ou em conjunto com outras primitivas é capaz de atender a necessidade expressa por um ou mais requisitos do sistema a ser modelado (e.g. estado e sinal). Um número excessivo de primitivas pode tornar o modelo complexo demais. Um número menor que o necessário pode eliminar a propriedade de *suficiência* do modelo (Seção 2.2.1). Assim, extrair as primitivas adequadas e suficientes é um desafio, pois nestas reside a base para a criação de um modelo de sistema computacional útil;
- ***Modelo computacional*** – é produzido por uma agregação específica de primitivas de modelagem, criando uma estrutura formal para representar uma classe significativa de sistemas computacionais;
- ***Modelo do sistema computacional*** – é uma representação abstrata do sistema computacional, realizada pela composição de um ou mais MOCs. Grande parte das linguagens de especificação de sistemas (e.g. C, Esterel, VHDL e SDL) provê suporte a vários MOCs e por este motivo, a maioria dos sistemas computacionais pode ser descrito por

qualquer linguagem de especificação. Todavia, a qualidade da modelagem dependerá fortemente do quanto as linguagens de especificação são adequadas para a representação do problema específico, ou seja, de como estas valorizam, através de suas sintaxe e semântica, a representação concreta de MOCs e primitivas de modelagem subjacentes ao problema.

2.2 Características de um MOC

2.2.1 Capacidade de Representação

As propriedades que mostram a capacidade de representação de um MOC são *suficiência* e *expressividade*, assim definidas:

- **Suficiência** – é a capacidade de um MOC de ser autônomo para representar todo o comportamento de um sistema computacional modelado através dele. Assume-se aqui que MOCs em geral são capazes de representar qualquer algoritmo computável, por terem poder computacional equivalente a máquinas de Turing;
- **Expressividade** – é a capacidade de um MOC para representar as estruturas básicas essenciais de um sistema computacional. A modelagem de um sistema deve ser realizada mediante o emprego de primitivas de modelagem suportadas pelo MOC escolhido. Caso as primitivas não sejam capazes de capturar as estruturas básicas do sistema computacional, o MOC é inadequado, isoladamente, para a modelagem em vista.

A capacidade de representação de um MOC é afetada pelo seu grau de especialização que reduz a quantidade de sistemas que podem ser expressos por este. Em contrapartida, quanto menos abstrato for este, mais expressivo ele será, aumentando a sua capacidade de dar melhor suporte à representação de diferentes sistemas. Em um extremo poder-se-ia assumir que cada sistema computacional teria o seu próprio MOC, em cujo caso os conceitos de MOC e modelo do sistema computacional confundem-se. Mas isto não é prático, pois existe uma infinidade de sistemas com características comuns que podem ser capturadas por MOC mais genérico, e tratadas uniformemente. Logo, a principal vantagem da genericidade é habilitar a automatização do processo de projeto com base nos MOCs, produzindo métodos e ferramentas automatizadas para aplicar no projeto de uma grande classe de sistemas computacionais. Apenas para exemplificar, compare-se dois circuitos de transmissão de dados, cuja diferença é uma palavra de alinhamento (seqüência ordenada de bits que permite sincronizar a operação do sistema). Ambos circuitos têm pelo menos uma diferença, o subcircuito que calcula o alinhamento. Todavia, o MOC subjacente não necessita em geral, ser alterado devido a este subcircuito. Em contrapartida, a arquitetura e o comportamento de um sistema não definem o MOC. Como exemplo, imagine um circuito que implemente uma determinada funcionalidade, e que este mesmo circuito deva ter sua freqüência de operação aumentada em 10 vezes. Neste caso, nem a arquitetura, nem o comportamento são alterados, mas devido às novas restrições de projeto, um circuito que poderia ser implementado por um MOC essencialmente seqüencial, poderá ter que ser implementado por um circuito com um MOC que explicite melhor o paralelismo e com isto atenda os requisitos de desempenho. O que se pode deduzir é que o MOC serve para refletir a organização[1] necessária para implementar o comportamento do sistema computacional frente ao seu conjunto de requisitos.

2.2.2 Interação com o Ambiente

Todo sistema computacional interage como o mundo externo a ele, ou seja, com seu *ambiente*. A interação com o ambiente relaciona-se a aspectos como o fluxo das informações processadas pelo sistema, sendo levados em consideração os modos de operação (síncrono ou assíncrono) em composição com os comportamentos (reativo, transformacional e interativo). As operações síncronas e assíncronas determinam diferentes modelos para permitir a comunicação entre os subsistemas (e.g. comunicação segura em sistemas assíncronos pode necessitar o emprego de filas). Uma classificação de sistemas segundo sua forma de interação com o ambiente é a seguinte [3]:

- **Sistemas reativos** – reagem a estímulos do ambiente gerando respostas a este. São sistemas voltados à entrada de informações com reação de acordo com o ritmo imposto pelo ambiente. Um modelo típico desta classe é o *Synchronous Reactive* (SR) [2][12].
- **Sistemas interativos** – interagem com o ambiente, porém, ao contrário dos sistemas reativos o gerador da interação é o sistema e não o ambiente. Neste caso, o sistema somente responderá a um estímulo quando estiver disponível.
- **Sistemas transformacionais** – são caracterizados pela baixa interação com o ambiente. Estes sistemas computam os valores de saída a partir dos valores de entrada e então param o processamento. Os modelos *Data Flow* (DF) [4][12][16][22] são típicos desta classe.

2.2.3 Composição de MOCs

A composição de MOCs é um critério de classificação do processo de modelagem. Neste sentido, pode-se classificar modelagem em dois grandes grupos:

- **Modelagem homogênea** – onde um único MOC tem suficiência e expressividade para representar o sistema computacional;

[1] *Organização* é uma das possíveis implementações de uma *arquitetura*.

- **Modelagem heterogênea** – onde é necessária a composição de mais de um MOC para a modelagem do sistema ter suficiência e expressividade adequadas.

Os sistemas de baixa complexidade ou com subsistemas com comportamentos semelhantes são bem representados por um único MOC. Quando aumenta a complexidade dos sistemas, os MOCs passam a não expressá-los mais adequadamente. Nestes casos, a abordagem heterogênea é uma solução.

2.3 Exemplos de MOCs

Vários são os MOCs encontrados na literatura [2][4][6] [12][13][18][19][22][23]. Estes são propostos de acordo com a interação do sistema com o ambiente, a forma de representação do tempo e a composição do modelo. Alguns exemplos de MOCs são descritos a seguir.

2.3.1 MOC Eventos Discretos

Uma grande quantidade de MOCs é composta por primitivas de tempo. O tempo pode ser modelado de forma discreta ou contínua. Os MOCs de tempo contínuo levam a informação de todos os sinais em todos os instantes de tempo, em contraposição aos MOCs de tempo discreto, onde somente os instantes de tempo que tem eventos associados (transições do valor de um sinal) são considerados. Nos MOCs discretos, um ou mais sinais são programados para ocorrer em instantes de tempos discretos. Um caso especial de MOC discreto é o modelo *Discrete Event* (DE) [12][13][22]. Neste MOC, a cada instante de tempo existe a referência a um único evento, tal como ilustrado na Figura 2. Nesta, embora o tempo seja contínuo, apenas nos instantes 2, 5, e 6 ocorreram eventos (as transições dos sinais A, B e C para 1, 0 e 1) e apenas um evento a cada instante.

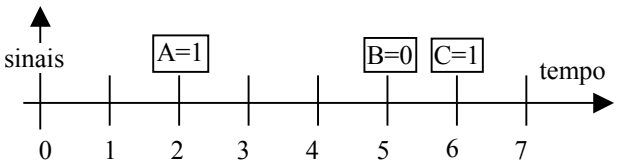

Figura 2 - Comportamento típico para o MOC DE.

O MOC DE é geralmente associado a linguagens seqüenciais. Cada processo é executado quando recebe um evento de entrada e produz um evento de saída, com tempo igual ou maior ao evento de entrada. Em caso de eventos simultâneos, o MOC DE se torna ambíguo. Para estes casos, existem diversas técnicas de solução da ambigüidade, tais como inserção de atrasos infinitesimais nos simuladores DE, o que acaba eliminando a simultaneidade dos eventos na modelagem. A grande vantagem deste MOC é a possibilidade de programar eventos futuros. Assim, no caso de uma simulação, o simulador não opera em instantes de tempo que não tem eventos associados, aumentando a velocidade da simulação.

No MOC DE, a comunicação ocorre de forma segura. Existem múltiplos escritores e um leitor. Os eventos são transportados globalmente e ordenadamente entre os processos [22].

2.3.2 MOC Máquina de Estados Finita

Em uma máquina de estados finita (em inglês, *Finite State Machine* ou FSM) o comportamento é especificado por um sistema de transições finito. Cada transição entre um par de estados contém uma relação de entradas que indica a transição e um conjunto de saídas produzidas pela transição.

FSMs são MOCs atraentes para sistemas embarcados [12], pois a quantidade de memória exigida pelo modelo é facilmente computada e é freqüentemente uma parte explícita da especificação. Questões relacionadas ao desempenho e término de execução também podem ser facilmente analisadas, já que cada estado pode, pelo menos em teoria, ser examinado em tempo finito.

Descrições com FSMs são comuns nos projetos de sistemas digitais, por sua capacidade de representar operações de controle complexas de forma compacta e clara. Todavia, este MOC não é adequado para representar modelos de memória e concorrência, devido ao problema da explosão combinatória do número de estados.

Para o caso de FSMs síncronas (SFSM), a comunicação é síncrona e não bloqueante, efetuada pelo compartilhamento de variáveis que são escritas e lidas em "tempo zero". Ambos, computação e comunicação acontecem instantaneamente em momentos discretos.

2.3.3 MOC Autômato Finito Não-Determinístico

Um autômato finito não-determinístico (em inglês *nondeterministic finite automata* ou NFA) é uma representação de um conjunto de estados finitos, com um único estado de início e um possível estado de fim, cujas transições entre estados são dadas por um único evento definido pelo alfabeto do autômato, e cujo conjunto de transições também é finito [9]. O autômato na sua definição básica não prevê um conjunto de saídas. Porém, extensões deste MOC que permitem uma ou mais saídas a cada transição podem ser úteis para a modelagem de sistemas computacionais.

O comportamento de um NFA é especificado de forma semelhante a um conjunto de grafos de transição, onde cada estado representa uma parcela de um processo de computação, os arcos representam seqüências totalmente ordenadas de eventos e o grafo representa um processo. Durante cada instante, os estados consomem eventos, executam ações e produzem eventos. O MOC NFA não representa o tempo explicitamente. Existe um comportamento determinístico na ordem parcial imposta

pela produção/consumo de eventos, mas os processos podem especificar comportamentos não-determinísticos através da descrição de estados que efetuam cálculos computacionais sem ter sido produzido um evento para a sua ativação.

Embora a ordem dos eventos seja determinística, a comunicação não é segura, dado que vários estados podem produzir eventos simultaneamente, mas cada estado pode consumir apenas um evento a cada instante. Para permitir que não haja perda de informação, o MOC prevê o uso de filas que armazenam os eventos na ordem de chegada dos processos e salva os eventos que não foram consumidos para estados futuros.

3 LINGUAGENS E MOCS SUBJACENTES

3.1 VHDL

VHDL (VHSIC (*Very High Speed Integrated Circuit*) *Hardware Description Language*) é uma linguagem padronizada pela IEEE, utilizada para descrever hardware em vários níveis de abstração [20]. VHDL foi originalmente utilizada para simulação e documentação, sendo posteriormente adotada para a síntese automatizada de sistemas digitais. O fato de ser um padrão aberto permitiu o intercâmbio de descrições entre várias ferramentas e sistemas de apoio ao projeto, fazendo com que se seja hoje muito usada em projetos de hardware.

3.1.1 Características e Modelo Subjacente

VHDL é uma linguagem imperativa com algumas características de orientação a objetos, de origem essencialmente paralela, mas que permite a programação seqüencial dentro de processos. Diferentes processos são usados para implementar comportamentos concorrentes.

VHDL é uma linguagem essencialmente síncrona, cujo modelo subjacente é o SFSM, embora os simuladores VHDL utilizem principalmente o modelo DE. O modelo SFSM não impossibilita VHDL de descrever sistemas assíncronos. VHDL também permite representar vários níveis de abstração, o que inclui desde o nível de chaves até o nível de sistemas [20].

A forma mais abstrata de descrever sistemas utilizando VHDL é através de construções comportamentais. Embora estas construções seja de grande valia para especificações iniciais e simulação destas, as ferramentas de síntese dificilmente conseguem obter bons resultados a partir de descrições com construções comportamentais, o que força os projetistas de VHDL terem em mente o hardware alvo, sempre que forem projetar um circuito e produzir eventualmente mais de uma descrição do projeto em mais de um nível de abstração. Embora VHDL tenha um alto potencial para a descrição de hardware, a linguagem não foi direcionada para a descrição de software, o que é fácil de notar pela incapacidade da linguagem para descrever operações dinâmicas, tais como alocação e liberação de memória, criação e destruição de processos.

3.1.2 Estrutura

VHDL foi projetada para permitir uma boa descrição hierárquica e modular dos circuitos. Para tanto, a construção principal é dada pelo par "entidade-arquitetura". Entidade é uma construção que define os sinais da interface de um bloco construído em VHDL com os demais blocos. Arquitetura é uma construção que descreve o comportamento do sistema, cuja interface está definida na entidade correspondente. Uma arquitetura é uma implementação de uma interface.

3.2 SDL

SDL (Specification and Description Language) é uma linguagem formal de descrição de sistemas digitais proposta, padronizada e mantida pela ITU (International Telecommunication Union) [14], desde 1976.

3.2.1 Características e Modelo Subjacente

SDL é uma linguagem imperativa, orientada a objetos e com representações textual e gráfica equivalentes. Para especificação, SDL é usada sobretudo na sua forma gráfica. A linguagem facilmente descreve a comunicação e a concorrência entre processos. O MOC subjacente aos processos descritos em SDL é baseado em uma extensão do NFA, que alguns denominam *rede de processos* [18], onde uma transição de estado implica no consumo[1] de um, e apenas um, evento[2] (que pode ser nulo). Desta forma processos devem ser descritos de forma a prever pelo menos um estado distinto para cada momento em que um dado sinal distinto deva ser consumido. Embora o MOC aparente seja o NFA estendido, a característica de permitir apenas uma transição por estado leva a uma ordenação total de eventos, que é típica do MOC DE.

Muito embora os estados descritos em SDL não estejam associados explicitamente ao tempo, a transição de estados não ocorre em tempo zero, o que impossibilita implementar uma cadeia de estados para consumir vários sinais que devam ser avaliados simultaneamente. Por este motivo, o modelo subjacente a SDL se destina principalmente para a descrição de sistemas assíncronos.

3.2.2 Estrutura da linguagem

SDL possui três componentes principais:

- **Sistema** – é o nível mais externo, o seu objetivo é delimitar o ambiente da especificação. O sistema

[1] O termo consumo é utilizado neste trabalho para referir o fato de SDL criar filas de eventos infinitas para a transição de um sinal. Sempre que um sinal é lido por um processo, ele é retirado das filas, ou seja, é consumido.

[2] Evento, no contexto de processos SDL, refere-se à transição de um sinal de entrada de um processo.

serve como um *recipiente* para o armazenamento de um conjunto de instâncias de blocos;

- **Blocos** – formam o nível intermediário da especificação, e têm por objetivo estruturar o sistema de forma clara, fornecendo um mecanismo de abstração gradativa da complexidade imposta pela especificação. Cada instância de um bloco é um *recipiente* para um conjunto de instâncias de processos ou para um conjunto de instâncias de outros blocos;
- **Processos** – realizam a camada mais próxima da implementação de um sistema, onde os sinais trocados dentro do sistema são efetivamente tratados. A instância de um processo ou é a implementação de uma máquina de estados, ou é um conjunto de instâncias de serviços.

O uso adequado dos componentes (sistema, bloco e processo) permite estruturar a especificação, reduzindo a complexidade do sistema. Uma descrição com estes componentes é essencialmente hierárquica, tal como pode ser observado na Figura 3.

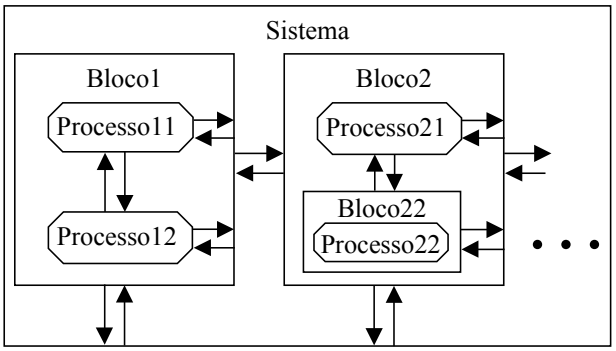

Figura 3 – Representação dos componentes SDL.

3.2.3 Obtenção de VHDL a partir de SDL

Devido à grande quantidade de ferramentas disponíveis no mercado para descrições em VHDL, alguns autores como Daveau e outros [10] dedicaram esforços para analisar a possibilidade e a facilidade de gerar descrições VHDL a partir de especificações SDL. Estas análises contribuíram para a geração de ferramentas de síntese automática de especificações SDL, tal como o Archimate [1]. Todavia, os autores deste trabalho concluem que devido à grande diferença entre os MOCs subjacentes, esta tarefa é muito onerosa, pois se torna muito difícil mapear descrições VHDL que geram um código sintetizável otimizado.

4 ESTUDO DE CASO

O estudo de caso usado para comparação entre MOCs de VHDL e SDL é um núcleo de propriedade intelectual (em inglês, *soft IP core*) de um sistema de telecomunicação que opera em tempo real. Este sistema, denominado DropInsert, tem como objetivo inserir e remover informações (dados ou voz) de quadros E1 [14].

O DropInsert é um circuito projetado para receber voz e dados de um quadro E1, inserir e retirar informações e retransmitir o novo quadro E1. A Figura 4 ilustra um diagrama de blocos do IP core do DropInsert, onde a interface com o ambiente é mostrada pelas flechas na periferia da Figura.

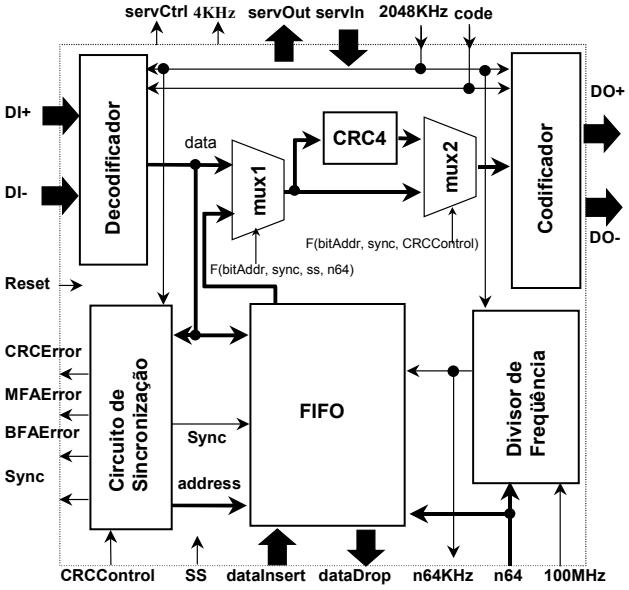

Figura 4 – Diagrama de blocos do IP core do DropInsert.

O DropInsert é composto pelos seguintes módulos:

- **Decodificador** – converte um sinal E1 diferencial, codificado em AMI ou HDB3 (sinais DI+ e DI-) em um sinal binário puro, que transporta o quadro E1;
- **Codificador** – converte a informação binária do quadro E1 de volta para um sinal diferencial (sinais DO+ e DO-);
- **Divisor de Freqüência** – gera várias freqüências necessárias às operações de inserção e remoção de informação do quadro E1;
- **Circuito de Sincronização** – realiza tarefas de alinhamento do quadro E1 e verificação de erros no protocolo de comunicação;
- **FIFO** – implementa as estruturas responsáveis por compatibilizar a freqüência das informações que estão sendo removidas e inseridas (sinais dataDrop e dataInsert) com a freqüência de operação do quadro;
- **Demais circuitos (CRC4 e circuitos auxiliares)** – realizam operações como calcular o CRC do novo quadro E1 (aquele formado pelo quadro anterior e

pelas informações inseridas) e inserir e remover informações de serviço (sinais servIn e servOut).

4.1 Características do DropInsert

O sistema DropInsert possui as seguintes características:

- Pequeno conjunto de subsistemas, de baixa complexidade algorítmica;
- Operação em tempo real;
- Necessidade de manipulação de bits, por implementar funcionalidades da camada de enlace do modelo OSI;
- Operação síncrona, embora faça parte de um sistema plesiócrono [14].

Estas características têm na verdade uma importância maior. Elas contêm o conjunto de requisitos essenciais para os quais o MOC, subjacente à modelagem, deve dar suporte. Ou seja, estas características direcionam para uma linguagem que tenha a melhor capacidade de especificar o sistema computacional. Neste exemplo, conclui-se que a característica de operação síncrona direciona para que o modelo deva ter primitivas que permitam explicitar o sincronismo. O pequeno conjunto de algoritmos de baixa complexidade mostra que a concorrência não é um requisito extremamente importante para esta especificação. A operação em tempo real direciona para modelos de hardware, onde se obtém maior velocidade de execução. Contudo, a característica de manipulação de bits influi mais diretamente na linguagem de descrição, através de tipos de dados e construções, do que no MOC subjacente.

4.2 Análise da Descrição em VHDL

A descrição do DropInsert em VHDL foi realizada de forma estrutural, presumindo uma implementação em um *hardware* síncrono. As construções entidade e arquitetura facilitam a tarefa de dividir a implementação em módulos. Bastou apenas definir precisamente cada interface e o projeto pôde ser desenvolvido de forma paralela sem a necessidade de muita interação entre os componentes da equipe. Todavia, nem todas as informações relevantes são fornecidas com a interface. Informações de como os processos operam com relação à lista de sensitividade podem ser bastante úteis. A falta desta informação pode ser verificada quando um sinal é transportado de uma arquitetura para outra, nas quais os processos para gerar e consumir os sinais, tinham a mesma borda de relógio e outra informação relativa a estes sinais era alterada na borda inversa. Neste caso, o segundo processo (que recebe os sinais do primeiro) tinha as informações atrasadas de um ciclo de relógio. Este tipo de problema foi verificado algumas vezes no projeto do DropInsert, implicando interação entre membros da equipe. Isto indica que a definição das interfaces VHDL dos módulos apenas não conduz a uma modularização perfeita do projeto.

A descrição VHDL é bastante adequada para o nível de descrição do DropInsert, pois ela permite manipular bits, avaliar sincronamente todos os sinais a partir da ocorrência de uma transição de um sinal de sincronismo. Além do mais, existe uma grande facilidade para descrever estados e transições de estado. Estas características levam os autores a concluir que VHDL é uma linguagem extremamente adequada para a descrição de sistemas computacionais com requisitos semelhantes ao DropInsert.

O grande problema encontrado para o projeto está associado à validação funcional do sistema como um todo, pois a quantidade de vetores de dados utilizada para a simulação representou muito tempo de projeto e dificulta a visualização de possíveis erros de especificação. Em resumo, descrições estruturais de sistemas complexos em VHDL apresentam níveis de abstração inadequados para validação inicial por simulação, apontando para a eventual utilidade de se desenvolver especificações iniciais comportamentais e validá-las via simulação antes de passar à descrição estrutural.

4.3 Análise da Descrição em SDL

A descrição do sistema DropInsert foi realizada de forma essencialmente hierárquica, com um conjunto de módulos semelhantes ao adotado na descrição VHDL.

A representação gráfica facilita a visualização e documentação das conexões entre os componentes de um sistema, tal como pode ser observado na Figura 5. Esta figura apresenta um bloco do sistema DropInsert, com os sinais e rotas de sinais entre 3 processos e entre o bloco e os demais blocos do sistema.

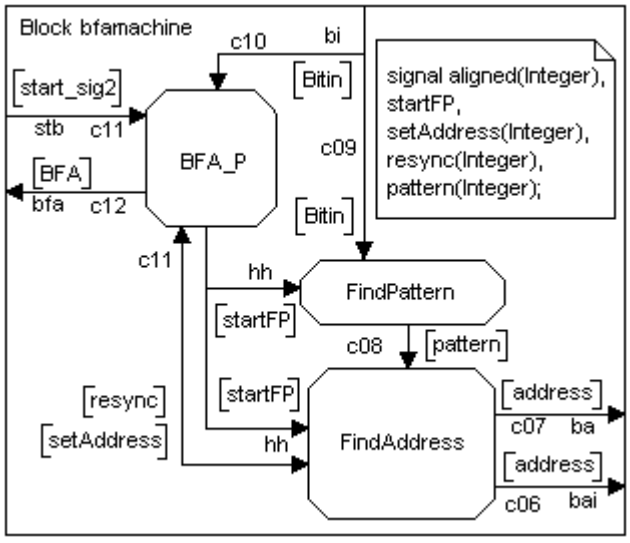

Figura 5 – Descrição em SDL de um bloco do DropInsert.

Uma vantagem observada na descrição do DropInsert está relacionada à curva de aprendizado devido ao emprego de descrição gráfica, ao pequeno número de componentes gráficos e à simplicidade da semântica dos mesmos. Em pouco tempo a equipe de projeto ambienta-se com a linguagem, o que leva o projeto a ser descrito e simulado num tempo menor que o equivalente em VHDL. Outra vantagem está associada à capacidade de documentação do arquivo fonte. A descrição gráfica SDL facilita a compreensão da estrutura do sistema devido, principalmente, à comunicação entre os componentes.

Todavia, conclui-se que a descrição SDL gráfica é adequada sobretudo para sistemas computacionais de baixa complexidade planar (aqueles com reduzido número de conexões entre blocos e processos). Para módulos de alta complexidade planar, as ferramentas visuais provocam um acúmulo demasiado de informações, tornando difícil a implementação e manutenção e reduzindo o potencial de documentação. Isto aponta que para controlar a complexidade de descrições gráficas, deva-se aumentar o número de níveis hierárquicos da descrição em relação a uma descrição textual, ou recorrer à forma textual de SDL.

Adicionalmente conclui-se que quando a complexidade planar aumenta, a velocidade de descrição dos projetistas reduz-se. Esta dificuldade também está associada à manutenção do projeto, pois algumas vezes a descrição tem que ser alterada e o tempo para remoção e inserção de blocos, processos e rotas de sinais é muito grande em comparação com uma linguagem textual.

Uma outra limitação importante encontrada na especificação SDL, através das ferramentas de descrição e simulação Cinderella [8], Tau [25] e ObjectGeode [24], é que a conexão de sinais em vários níveis da hierarquia do sistema deve ser realizada com o mesmo nome. Isto dificulta a modularidade da descrição dos blocos e processos, pois supondo que tenham sido implementados projetos distintos, cada qual com parte da funcionalidade do sistema e com nomes de sinais não padronizados, ao unir estes projetos em um único, as ferramentas de descrição ou síntese geram erro, pois não inferem que um sinal de nome diferente pode ser transportado por uma rota de sinal e ser renomeado ao passar para um outro nível hierárquico. Este característica mostra que, embora as versões de SDL92 em diante sejam orientadas a objeto, existe uma dificuldade natural de reuso de módulos.

Contudo, o maior problema encontrado para descrever o DropInsert em SDL está associado ao MOC subjacente. O fato da linguagem presumir a implementação da recepção de eventos através de filas dificulta a descrição de sistemas onde mais de um sinal deve ser analisado em um mesmo instante de tempo. A solução adotada em SDL é prover uma construção para "salvar" um ou mais eventos que seriam analisados em paralelo num mesmo estado para análise em estados posteriores, que são adicionados artificialmente à descrição. As implementações desta construção é bastante complexa e em geral as ferramentas de síntese não geram código para a mesma (e.g. Archimate [1]). A própria ITU-T, em [15], sugere que as descrições SDL sejam realizadas sem esta construção. Como solução alternativa, os trechos de descrição que implicam na leitura de mais de um sinal devem ser seqüencializados ou deve ser criado algum mecanismo de salvamento explícito.

Na Figura 6 e na Figura 7 aparecem as descrições do processo FindPattern em VHDL e SDL, respectivamente. O leitor pode analisar a descrição VHDL e verificar que quando ocorre um evento de relógio (sinal clock2048), as combinações de valores de dois sinais de entrada (pattern e BitIn) são relevantes para determinar o comportamento posterior do processo. Na descrição SDL, foram criados dois estados (S0 e S1) que têm por objetivo consumir as entradas do processo. Devido ao fato de que não é garantido que o consumo dos eventos ocorra na mesma ordem em que estes são gerados, esta descrição apresenta um outro problema potencial, que é a grande dificuldade de representação de comportamentos síncronos.

```
FindPattern: process(startFP, clock2048)
begin
  if(startFP='1') then
    aligned <= '0';
    pattern <= (others => '1'); -- pattern<=127
  elsif(clock2048'event and clock2048='0') then
    if(pattern="0011011") then  -- pattern=27
      aligned <= '1';
    else
      aligned <= '0';
    end if;
    pattern(6 downto 1) <= pattern(5 downto 0);
    pattern(0)<= BitIn;
  end if;
end process FindPattern;
```

Figura 6 - Descrição do processo FindPattern em VHDL.

O processo FindPattern descrito em SDL tem como estado inicial S0, onde se permanece enquanto não ocorrer um evento no sinal BitIn. Como não é garantida a ordem dos eventos BitIn e pattern, caso um evento do sinal pattern ocorra antes do evento do sinal BitIn, este evento será salvo usando a construção “save”, representada graficamente por um paralelogramo contendo o nome do sinal pattern. Uma vez recebido o evento BitIn, o processo vai para o estado S1, onde permanece aguardando um evento no sinal pattern (que pode ter ocorrido no estado anterior). Ao ocorrer o evento do sinal pattern, o valor recebido é armazenado na variável patternAux. O bit menos significativo de patternAux recebe o valor do sinal BitIn (armazenado na variável temp) e patternAux é deslocado um bit à esquerda. O resultado da operação, módulo 128, é transmitido para fora do processo através do sinal pattern. Caso o valor de patternAux seja 27 é gerado o sinal aligned. Independente de encontrar o valor 27, o processo volta ao estado S0, para aguardar um novo evento BitIn. Concorrentemente aos estados S0 e S1,

existe um trecho de código encabeçado por um símbolo de estado contendo o símbolo '*'. Isto indica que sempre que ocorrer um evento do sinal startFP, independente de qual o estado atual (S0 ou S1), o sinal pattern receberá o valor 127 e o processo irá para o estado S0. Este procedimento permite simular a inicialização do processo.

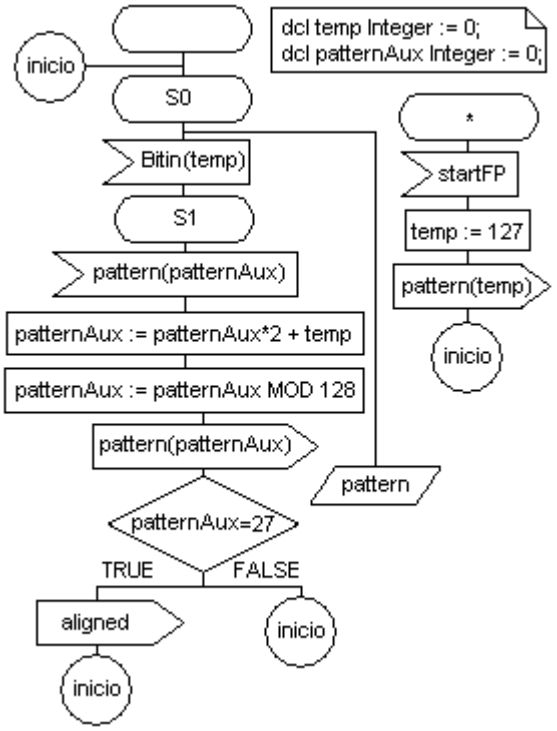

Figura 7 - Descrição do processo FindPattern em SDL.

Uma vantagem da descrição SDL em relação à descrição CHDL, que pode ser observada através da comparação entre a Figura 6 e a Figura 7, é o maior nível de abstração. Por exemplo, **aligned** e **startFP** são sinais puros em SDL (não valorados), o que importa é apenas a sua presença ou ausência. Além do mais, a descrição SDL permite abstrair o sinal **clock2048**, avaliando-se apenas a presença de evento no sinal **BitIn**.

4.4 Comparação e Resultados

As descrições SDL e VHDL foram comparadas de acordo com uma série de critérios objetivos e subjetivos.

4.4.1 Critérios Objetivos

O conjunto de critérios objetivos é composto por elementos palpáveis embora dependentes da habilidade de descrição por parte dos projetistas. Nestes critérios, resumidos na Tabela 1, o que mais se destaca é o tempo total de projeto que foi consideravelmente reduzido com SDL. Todavia, deve ser levado em consideração que o projeto elaborado em SDL foi realizado após a especificação VHDL e que um dos componentes do grupo que participou da especificação em SDL também participou da especificação em VHDL.

Tabela 1 – Comparação de critérios objetivos (VHDL e SDL).

Critérios	VHDL	SDL
Número de pessoas (15 horas/semana)	5	2
Tempo de projeto (meses)	4	2
Tempo total de projeto (horas)	1.200	240
Número de módulos/blocos	12	17
Número de processos	35	39
Número de linhas de código/símbolos gráficos	1800	2114

4.4.2 Critérios Subjetivos

Os critérios subjetivos servem como indicadores qualitativos. Estes critérios, resumidos na Tabela 2, mostram que SDL obtém melhores resultados com relação a curva de aprendizado, capacidade de documentação e abstração da linguagem, enquanto que VHDL mostra-se mais flexível para alterações de especificação, tem uma grande facilidade para a geração de testes da descrição e, principalmente, tem um modelo subjacente mais adequado para a especificação do sistema computacional DropInsert. Adicionalmente, VHDL também dá melhor suporte ao reuso de módulos que SDL

Tabela 2 – Comparação de critérios subjetivos (VHDL e SDL).

Critérios	VHDL	SDL
Curva de aprendizado	alta	baixa
Capacidade de documentação	média	alta
Nível de abstração da descrição	baixo	alto
Facilidade de simulação (*test bench*)	alta	média
Flexibilidade de alteração da descrição	alta	média
Reutilização de código	média	baixa
Adequação ao modelo computacional empregado na modelagem	alta	média

5 CONCLUSÕES E TRABALHOS FUTUROS

O DropInsert mostrou uma comparação entre duas linguagens com MOCs distintos para a especificação de um sistema de telecomunicações. Devido aos requisitos deste sistema, principalmente ao requisito de sincronismo, a linguagem VHDL, que tem suporte ao modelo SFSM, é mais adequada para expressar as necessidades do sistema. Já a linguagem SDL se torna inadequada, pois o modelo básico subjacente é uma extensão do NFA.

O conjunto de critérios objetivos e subjetivos leva a conclusões sobre vantagens e desvantagens de cada uma das linguagens para a especificação de sistemas. entre estas, conta-se como vantagens de SDL a curva de aprendizado, a capacidade de documentação e abstração.

Em contrapartida a linguagem VHDL é ser mais flexível à manutenção da descrição e mais adequada para a realização de testes da descrição.

A descrição VHDL foi completamente verificada através de simulação, implementada fisicamente e teve verificada sua funcionalidade em um protótipo em FPGAs. A descrição SDL foi completamente especificada e verificada através de simulação e está em fase de implementação através da ferramenta Archimate. O que se deseja, para trabalhos futuros, é efetuar comparações em relação à implementação física, gerando implementações em hardware, software e mistas, e relacionar com os resultados obtidos através de VHDL.

REFERÊNCIAS

[1] AREXSYS – **Archimate User Manual. Arexsys Inc**. Available at: http://www.arexsys.com, 2001.

[2] A. BENVENISTE and G. BERRY – The Synchronous Approach to Reactive and Real-Time Systems. **Proceedings of the IEEE**, vol. 79, no. 9, pp. 1270-82, September 1991.

[3] G. BERRY – **The Esterel v5 Language Primer Version v5_91.** Available at: http://www.esterel.org, INRIA, July 2000.

[4] B. BHATTACHARYA and S. S. BHATTACHARYYA – Parameterized dataflow modeling for DSP systems. **IEEE Trans. on Signal Processing**, vol. 49, pp. 2408-2421, Oct. 2001.

[5] G. V. BOCHMANN – Specification Languages for Communication Protocols. **Proceedings of the Conference on Hardware Description Languages**, April 1993.

[6] J. BUCK and R. VAIDYANATHAN – Heterogeneous modeling and simulation of embedded systems in El Greco. **CODES 2000**, pp. 142-146, May 2000.

[7] L. CAI, D. GAJSKI, M. OLIVAREZ - Introduction of system level architecture exploration using the SpecC methodology**. IEEE International Symposium on Circuits and Systems**, pp: 9–12, vol. 5, May 2001.

[8] CINDERELLA. **A visual SDL tools for systems development**. Available at homepage: http://www.cinderella.dk, 2001.

[9] D. I. A. COHEN – **Introduction to Computer Theory, Second Edition**, John Wiley & Sons Inc, 1997.

[10] J. M. DAVEAU, G. F. MARCHIORO, C. VALDERRAMA and A. A. JERRAYA – **VHDL generation from SDL specification**, Chapman & Hall, 1996.

[11] G. ECONOMAKOS, P. OIKONOMAKOS, I. PANAGOPOULOS, I. POULAKIS, G. PAPAKONSTANTINOU – Behavioral synthesis with SystemC. **Design, Automation and Test in Europe**, pp. 21 – 25, March 2001.

[12] S. EDWARDS, L. LAVAGNO, E. LEE and A. SANGIOVANNI-VINCENTELLI – Design of Embedded Systems: Formal Models, Validation and Synthesis. **Proceedings of the IEEE**, vol. 85, no. 3, March 1997.

[13] A. GIRAULT, B. LEE and E. LEE – Hierarchical Finite State Machines with Multiple Concurrency Models. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, vol. 18, no. 6, June 1999.

[14] ITU. **CCITT Specification and Description Language (SDL)**. ITU-T Recommendation Z.100, March 1993.

[15] ITU. **SDL Methodology Guidelines, SDL Bibliography**. ITU-T Recommendation Z.100 – Appendices I and II. March 1993.

[16] A. A. JERRAYA, M. ROMDHANI, PH LE MARREC, F. HESSEL, P. COSTE, C. VALDERRAMA, G. F. MARCHIORO, J. M. DAVEAU and N.-E. SERGAINOH – **System-Level Synthesis (NATO Science Series)**, chapter: Multilanguage Specification for System Design, pp.103-136, Kluwer Academic Publishers, 1999.

[17] T. KUHN, W. ROSENTIEL and U. KEBSCHULL – Description and Simulation of Hardware/Software Systems with Java. **Design Automation Conference**, New Orleans, 1999.

[18] L. LAVAGNO, A. SANGIOVANNI-VICENTELLI and H. HSIEH – **Embedded System Codesign: Synthesis and Verification**. Proceedings of the NATO Advanced Study Institute in Hardware/Software Co-Design, Kluwer Academic Publishers, Italy, 1995.

[19] B. LEE and E. A. LEE – Interaction of Finite State Machines and Concurrency Models. **Conference Record of the Thirty-Second Asilomar Conferece on Signals, Systems & Computers**, vol. 2, pp. 1715-19, 1998.

[20] D. PERRY - **VHDL 3rd Ed.**, McGraw-Hill, New York, 1998.

[21] D. SCIUTO - Guest Editor's Introduction: Design Tools for Embedded Systems.**IEEE Design & Test of Computers**, Apr.2000.

[22] M. SGROI, L. LAVAGNO and A. SANGIOVANNI-VINCENTELLI – Formal Models for Embedded System Design. **IEEE Design & Test of Computers**, April 2000.

[23] V. SKLYAROV – Hierarchical Finite-State Machines and Their Use for Digital Control. **IEEE Transactions on VLSI Systems**, vol. 7, no. 2, June 1999.

[24] TELELOGIC OBJECTGEODE **- A SDL tools for real time systems development**. Telelogic, Inc. Available at homepage: http://www.telelogic.com, 2001.

[25] TELELOGIC TAU. **A SDL tools for real time systems development**. Telelogic, Inc. Available at homepage: http://www.telelogic.com, 2001.