Centro Científico Rio Relatório Técnico CCR-136

Gerência de Metodologias de Projeto no Ambiente GARDEN

por

Flávio Rech Wagner

IBM Brasil - Centro Científico Rio
Av. Presidente Vargas 824/844
20.071, Rio de Janeiro - RJ

Sistemas digitais - SBJ/II
Ambiente: Projeto
GARDEN

CNPq 3.04.05.00-9

Sumário

Este relatório discute os fundamentos de um modelo para a gerência de metodologias de projeto de sistemas e circuitos eletrônicos no ambiente GARDEN. Na abordagem proposta, a gerência de metodologias de projeto em GARDEN está intimamente relacionada ao modelo de dados suportado pelo ambiente, que oferece uma grande flexibilidade na definição de esquemas conceituais para diferentes aplicações. A especificação de uma metodologia de projeto define simultaneamente um esquema particular. A característica mais saliente e inovadora do método proposto é a possibilidade de serem definidas hierarquias de metodologias de projeto. O relatório não descreve a ferramenta que implementa a abordagem proposta. O relatório compara o conceito de metodologia de projeto no escopo do ambiente GARDEN com conceitos definidos em outros ambientes.

Abstract

This report discusses the foundations of a model for design methodology management for electronic circuits and systems in the GARDEN design framework. According to this approach, design methodology management in GARDEN is intimately related to the data model that is supported by the framework. This model offers a great flexibility in the definition of conceptual schemes for different applications. The specification of a design methodology simultaneously defines a particular scheme. The most remarkable and innovative feature of the proposed method is the possibility of defining hierarchies of design methodologies. The report does not describe the tool which implements the proposed approach. The report compares the concept of design methodology within the scope of the GARDEN framework with concepts defined in other systems.

O autor realizou este trabalho durante licença da Universidade Federal do Rio Grande do Sul – Instituto de Informática e Curso de Pós-Graduação em Ciência da Computação

Conteúdo

Lista de Figuras

1 Introdução

Ambientes de projeto de sistemas e circuitos eletrônicos objetivam a integração de ferramentas de projeto de uma maneira a obter consistência de dados e uniformidade na interação com o usuário e a auxiliar projetistas na gerência de dados e de metodologias de projeto.

A característica principal de um ambiente de projeto é o suporte a um modelo de dados uniforme [1], que permite a representação de sistemas e circuitos como objetos complexos, considerando aspectos como composição e hierarquia, e que oferece recursos para a gerência de dados, incluindo facilidades como múltiplas representações para um objeto e controle de versões e configurações.

Um modelo de dados com estas características foi definido para GARDEN [2], um ambiente aberto à integração de ferramentas (um *framework*), em especificação no Centro Científico Rio da IBM Brasil. GARDEN oferece um modelo de dados bastante flexível, de modo a permitir a construção de ambientes particulares (*environments*), dedicados a diferentes arquiteturas, tecnologias e/ou metodologias de projeto.

No atual estágio do projeto GARDEN, ainda não foram detalhados mecanismos para outros aspectos importantes de ambientes, tais como gerência de metodologias de projeto, integração de ferramentas, comunicação entre ferramentas e interface com o usuário.

Este relatório propõe uma abordagem para a gerência de metodologias de projeto em GARDEN. Como se verá, nesta abordagem a gerência de metodologias de projeto está intimamente relacionada à integração de ferramentas e à definição de esquemas conceituais para aplicações particulares, em função dos princípios que nortearam a definição do modelo de dados do GARDEN. Não é intenção deste relatório, no entanto, detalhar as ferramentas que implementam o modelo proposto.

O relatório está organizado da maneira que se segue. Na seção 2 é apresentado o modelo de dados do ambiente GARDEN, já suficientemente explorado em trabalhos anteriores [1,2,3], mas aqui incluído para manter a consistência interna do relatório. Na seção 3 são discutidos mapeamentos de outros modelos de dados para o ambiente GARDEN, estendendo idéias originalmente sugeridas em [4]. Esta discussão é importante para a compreensão da flexibilidade oferecida pelo modelo GARDEN e das possibilidades de definição de ambientes que ele oferece. Na seção 4 é então apresentada a proposta do modelo de gerência de metodologias de projeto, que também contempla os aspectos de integração de ferramentas e de definição de esquemas conceituais para aplicações particulares. Seguem-se, na seção 5, exemplos de utilização do modelo proposto. Na seção 6 é feita uma comparação do conceito e dos mecanismos de gerência de metodologias de projeto definidos no âmbito do projeto GARDEN com aqueles utilizados em outros ambientes. Na seção 7, finalmente, são resumidas as contribuições mais importantes deste trabalho e são discutidos trabalhos futuros a serem realizados a partir da presente proposta.

2 Modelo de dados GARDEN

O modelo de dados do ambiente GARDEN [2,4] [1], em desenvolvimento no Centro Científico Rio da IBM, tem por objetivos suportar a modelagem de objetos complexos e a gerência das diversas representações de um objeto criadas ao longo do processo de projeto, de uma maneira flexível e independente de aplicações particulares. O ambiente é configurável para diferentes tecnologias de fabricação, servindo tanto ao projeto de circuitos VLSI como de sistemas compostos por componentes discretos, e mesmo à integração completa de sistemas, considerando o projeto físico de placas e processos de encapsulamento e montagem de componentes.

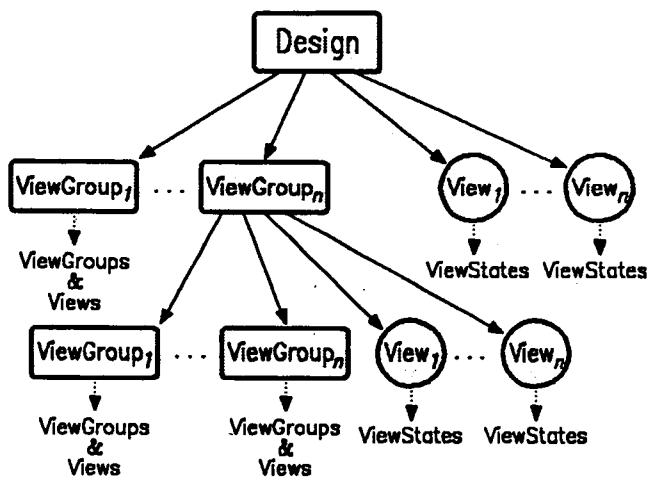


Figura 1: Objetos básicos do modelo de dados do GARDEN

O núcleo deste modelo de dados é mostrado na Figura 1 [2]. Sua estrutura básica oferece uma hierarquia para a representação e gerência de objetos complexos, composta pelos conceitos **Design**, **ViewGroup**, **View**, **Modificações** e **Iterações**, estes dois últimos formando **ViewStates**. ViewGroups podem conter outros ViewGroups ou Views, podendo ser criada assim uma representação hierárquica com profundidade qualquer. Esta representação será denominada neste trabalho de **estrutura de controle** de um objeto (um módulo ou sistema).

Views podem ser de três tipos: **HDL**, **Layout** e **MMHD** ("Mixed-Mode Hierarchical Description"). Todos os tipos de Views podem conter componentes, que fazem referência a outros Designs, mas apenas Views MMHD representam as interconexões entre sinais de interface dos componentes. A referência pode ser feita a qualquer nível da estrutura de controle destes outros Designs. É possivel fazer-se

[1] Esta descrição do modelo de dados do ambiente GARDEN foi extraída de [1]

[2] A Figura 1 foi extraída de [4]

uma referência a um ViewState particular de uma View ou uma referência genérica ao ViewState mais recente.

Conforme ilustrado na Figura 2 [3], Modificações são descrições alternativas para uma determinada View, enquanto Iterações são refinamentos sucessivos de uma mesma Modificação. Na notação $i.j$, i identifica a Modificação, enquanto j identifica a Iteração.

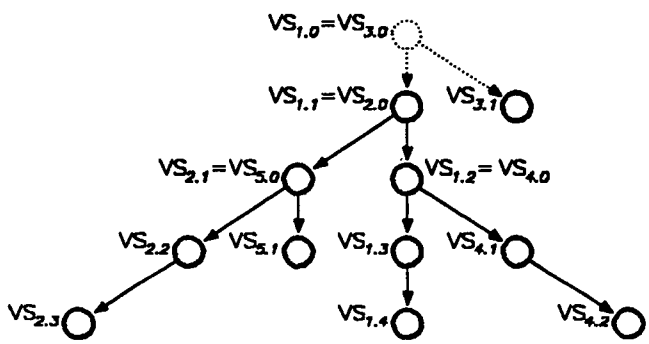


Figura 2: Modelo de dados do GARDEN – modificações e iterações

Sinais de interface (**Ports**) podem ser definidos nos níveis de Design, ViewGroup e View, sendo herdados pelos níveis inferiores da hierarquia. Com isto é possível a representação de portas apenas nos níveis de abstração nos quais elas realmente têm um significado. Além disto, é possivel representar-se decisões de projeto alternativas que herdam portas comuns mas acrescentam portas específicas.

Além de portas, o usuário pode definir outros **Atributos** para um nodo N_i qualquer da estrutura de controle. Estes Atributos são automaticamente herdados por todos os nodos descendentes de N_i. Atributos têm um nome e seu valor pode ser um *string* ou estar armazenado em um arquivo cujo conteúdo não é detalhado no modelo de dados.

Correlações podem ser estabelecidas entre dois ou mais objetos quaisquer, em qualquer nível de suas respectivas estruturas de controle. Correlações podem representar relações de equivalência entre ViewGroups e/ou Views e/ou ViewStates de um mesmo Design. Pode-se também representar o fato de que dois ou mais objetos, p.ex. o ViewGroup VG_i do Design *DataPath* e o ViewGroup VG_j do Design *ControlUnit*, correspondem a uma mesma decisão de projeto (p.ex. relógio de 4 fases), ou ainda que um objeto, p.ex. uma View MMHD V_k de um Design D_n, composta de portas lógicas, foi obtida acrescentando-se informações de *timing* a outra View MMHD V_l de D_n (processo conhecido por *back-annotation*), informações estas extraídas de outra View Layout V_m de D_n.

O banco de dados do GARDEN é denominado de **Repositório**, e contém **Bibliotecas** e definições de **Processos**. Bibliotecas agrupam Designs segundo critérios

[3] A Figura 2 foi extraida de [4]

estabelecidos pelos usuários. Um Processo consiste basicamente de um arquivo contendo informações sobre uma determinada tecnologia de fabricação. Uma Biblioteca pode ter um Processo associado a ela, caso em que este Processo será aquele no qual serão implementados todos os Designs existentes na Biblioteca. Um Processo P_i pode ter outros Processos nele **incluíveis**. Estes Processos incluídos definem tecnologias compatíveis com a tecnologia de P_i. Uma ferramenta de configuração poderia restringir referências dentro de uma View de um Design D_j, contido numa Biblioteca com Processo P_i, a outros Designs contidos em Bibliotecas cujos Processos estão incluídos em P_i.

Um Processo pode alternativamente ser associado a Designs, ViewGroups ou Views, de forma individual. Isto é possível quando o objeto (Design, ViewGroup ou View) estiver contido em outro ao qual não está associado um Processo. Pode-se por exemplo associar um Processo a um ViewGroup desde que o Design ao qual este ViewGroup pertença não tenha Processo associado diretamente a si ou à sua Biblioteca.

TimeStamps permitem o controle da evolução de diversos atributos no Repositório. Atributos que podem ser controlados desta forma têm armazenada no Repositório uma história de alterações, numa forma linear. Num tempo qualquer, uma referência a um objeto pode ser feita à sua versão corrente, com o que são buscados os valores de atributos para o tempo corrente (aqueles com maior valor de TimeStamp), ou a uma versão passada, com o que são buscados valores de atributos com TimeStamp mais próximo do (mas menor do que o) tempo no qual se quer fazer a referência. Pode-se recuperar configurações passadas de objetos, já que referências em configurações dinâmicas são também atributos temporais. Também o nome do Processo associado a uma dada Biblioteca ou Design é um atributo temporal, permitindo-se com isto um mecanismo de configuração inteiramente controlado pelo tempo (lembrar relação entre Processo e configuração aludida no parágrafo anterior).

O modelo de dados do GARDEN é bastante flexível, já que ViewGroups têm uma semântica aberta. O critério de agrupamento de Views (ou outros ViewGroups) sob um ViewGroup pode ser definido pelo usuário e a hierarquia de ViewGroups pode ter uma profundidade qualquer, com o que pode-se criar na prática diferentes estruturas de controle. Em [3], por exemplo, é simulado um processo de projeto que resulta na criação de objetos GARDEN atendendo a um critério pelo qual cada ViewGroup é o agrupador de descrições que resultam de uma mesma decisão de projeto, conforme parcialmente ilustrado na Figura 3. Estes ViewGroups estão organizados hierarquicamente, de um modo que reflete a seqüência de níveis de abstração pela qual passa o projeto. Neste esquema, portas vão sendo acrescentadas à interface de um objeto à medida que decisões de projeto são tomadas.

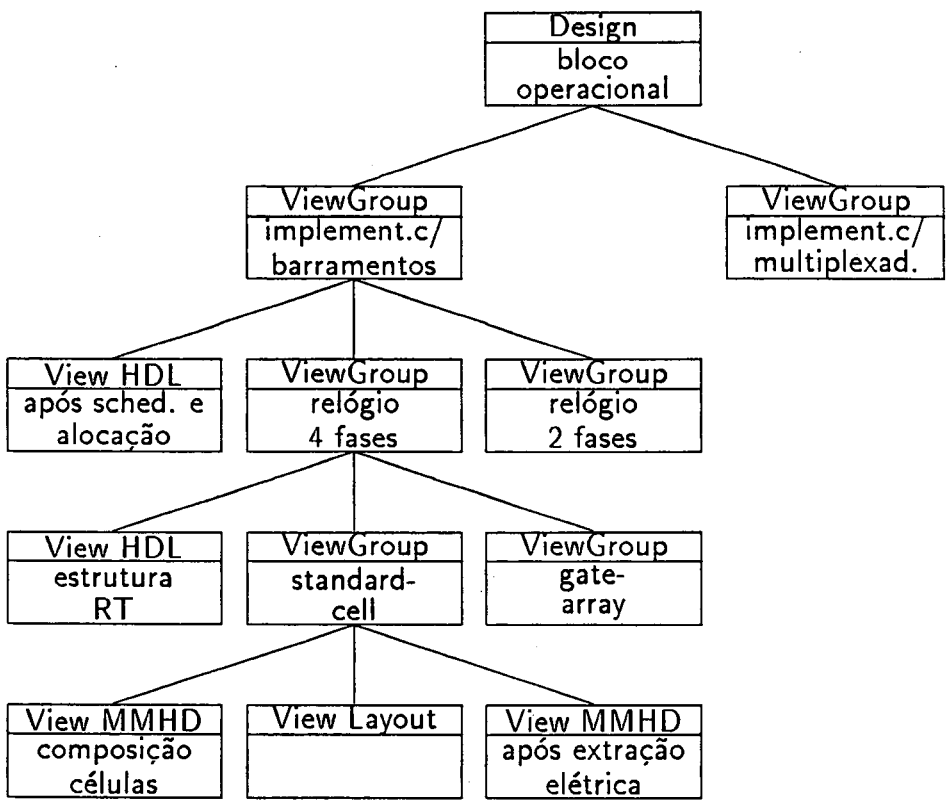


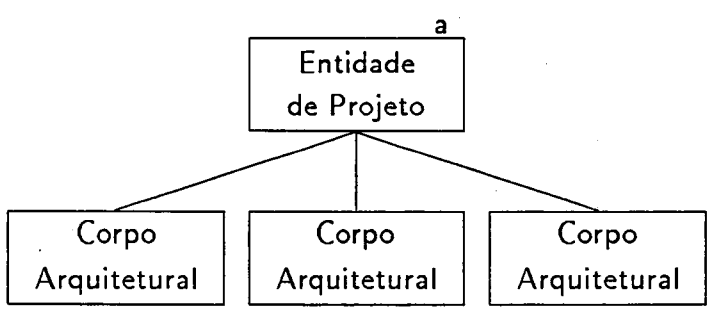
Figura 3: Exemplo de estrutura de controle implementada em GARDEN

3 Mapeamento de outros modelos de dados para o GARDEN

Nesta seção é mostrado de que maneira poderiam ser mapeados para o GARDEN os modelos de dados dos sistemas Oct [5] e DAMASCUS [6] e da linguagem VHDL [7], conforme originalmente sugerido em [4]. Em [1] são discutidas as vantagens relativas destes vários modelos, o que não é objetivo do presente relatório. Será visto que o mapeamento apresenta em cada caso pequenas restrições em função de propriedades particulares dos outros modelos não encontráveis em GARDEN. Estas restrições não comprometem no entanto a retenção pelo mapeamento das propriedades mais importantes dos diversos modelos. Como se verá posteriormente, o modelo de gerência de metodologias de projeto proposto neste trabalho está baseado justamente na flexibilidade de definição de diferentes modelos de dados a partir do modelo básico do GARDEN.

3.1 VHDL

VHDL [7] é uma linguagem de descrição de hardware desenvolvida sob um contrato do DoD e adotada como padrão pela IEEE. Muitos produtos comerciais suportando a linguagem estão disponíveis. Embora VHDL não seja um ambiente, mas uma linguagem que pode ser integrada em um ambiente, ela já apresenta um esquema de gerenciamento das representações de objetos. Este esquema pode ser estendido por ferramentas externas, mas impõe algumas restrições conceituais que não podem ser evitadas.



a. com declaração de interface

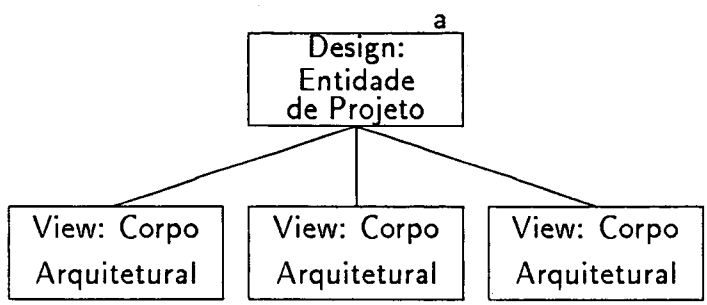
Figura 4: Estrutura de controle da linguagem VHDL

Objetos de projeto em VHDL são modelados como **Entidades de Projeto**, que são descritas através de uma **interface** e um ou mais **Corpos Arquiteturais**, correspondendo a diferentes representações para a Entidade, conforme ilustrado na Figura 4. Três diferentes estilos de descrição podem ser utilizados dentro de um

mesmo Corpo Arquitetural: **comportamental**, **dataflow** e **estrutural**. Neste último estilo, o Corpo é descrito como uma interconexão de **componentes**.

Componentes podem ser ligados a uma determinada Entidade através de uma especificação de configuração dentro do Corpo que os contém ou através de um **Corpo Configuracional** separado. Um Corpo Arquitetural pode ser **parcialmente ligado**, se apenas Entidades são selecionadas para seus componentes, **completamente ligado**, se também Corpos Arquiteturais destas Entidades são selecionados, ou **abertos**, se a ligação é feita separadamente através de um Corpo Configuracional.

Uma estrutura de controle semelhante à de VHDL pode ser obtida em GARDEN simplesmente conectando-se as Views (que corresponderão aos Corpos Arquiteturais) diretamente à raíz do Design (que corresponderá a uma Entidade), sem a utilização de ViewGroups, conforme mostrado na Figura 5. A definição da interface é feita completamente no nível do Design. Como em VHDL não há diferentes tipos de Corpos Arquiteturais, todos os Corpos são mapeados para Views de tipo HDL (que, assim como os Corpos Arquiteturais, podem conter componentes que fazem referência a outras Entidades / Corpos Arquiteturais). Configurações estáticas de VHDL correspondem a referências a Views em GARDEN, enquanto configurações dinâmicas correspondem a referências a Designs.



a. com declaração de interface

Figura 5: Estrutura de controle da linguagem VHDL mapeada para GARDEN

O mapeamento de VHDL para GARDEN apresenta três restrições:

- em GARDEN não se pode declarar configurações abertas;
- em GARDEN não se pode armazenar configurações como nos Corpos Configuracionais de VHDL;
- em VHDL é possível que um componente utilize menos portas do que as declaradas na Entidade que ele referencia, o que não pode ser feito em GARDEN.

3.2 Oct

A Figura 6 mostra os objetos básicos manipulados pelo gerenciador de dados Oct [5], desenvolvido pela Universidade de Berkeley. **Células** podem conter qualquer número de **Views**. Não há um conjunto pré-definido de tipos de Views. Cada View tem um número qualquer de **Facetas**. Uma Faceta especial, denominada **Contents Facet**, contém a descrição completa da estrutura e geometria internas da View, além da definição de sua interface básica. As restantes Facetas de uma View, denominadas **Interface Facets**, herdam esta interface básica e acrescentam atributos adicionais, de modo a implementar *protection frames*, que definem informações visíveis externamente à View e dedicadas a ferramentas especializadas. Exemplos são a definição geométrica da envoltória da Célula, para um editor de esquemáticos, regiões de roteamento (tanto ao longo da interface como regiões de transparência sobre a Célula) para um roteador, e a geometria que precisa ser verificada contra as Células vizinhas, para um DRC. Cada Faceta pode ter várias **Versões**. Componentes em descrições compostas fazem referência a Views de Células.

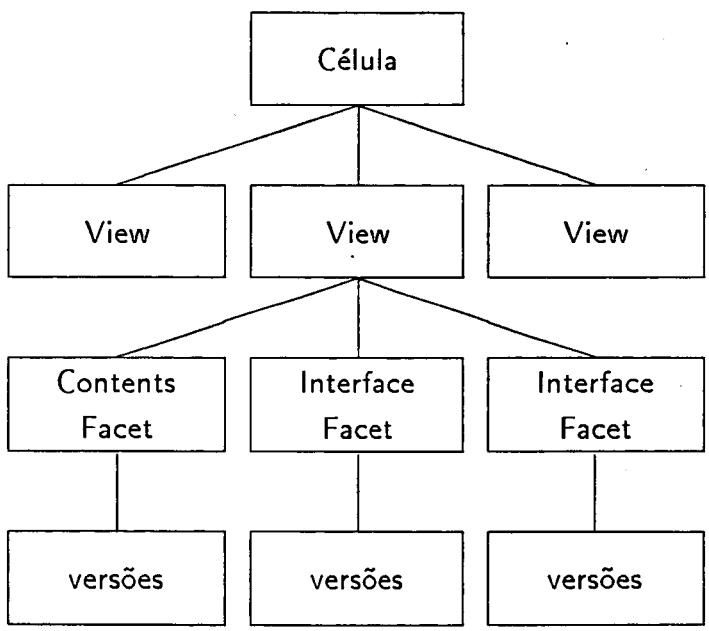
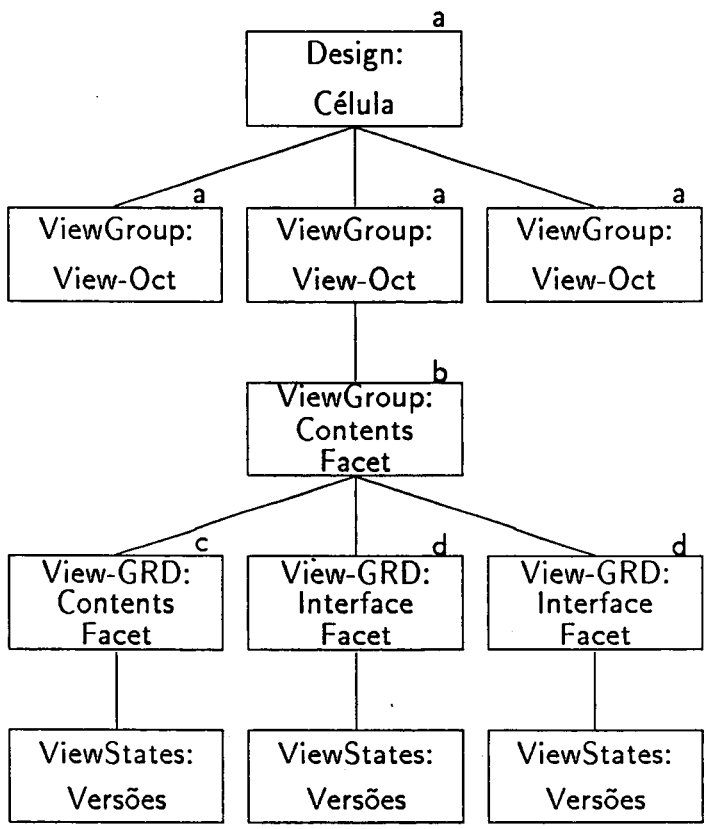


Figura 6: Estrutura de controle do gerenciador Oct

Uma estrutura de controle baseada no modelo Oct é implementada em GARDEN como mostrado na Figura 7. Células Oct correspondem a Designs GARDEN. Uma View V_o de Oct é mapeada em um ViewGroup VG_{g1} de GARDEN. A Contents Facet de V_o é mapeada em um ViewGroup VG_{g2} sob VG_{g1} e em uma View V_g sob este VG_{g2}. VG_{g2} armazena a definição da interface básica do objeto, enquanto V_g armazena o conteúdo da Faceta. Outras Interface Facets desta View Oct V_o são Views GARDEN sob VG_{g2}, que acrescentam atributos visíveis externamente à

View.



a. sem declaração de interface
b. com declaração de interface
c. com o conteúdo real da Contents Facet
d. com atributos adicionais que são externamente visíveis

Figura 7: Mapeamento da estrutura de controle Oct para GARDEN

3.3 DAMASCUS

A Figura 8 [4] mostra a estrutura de controle de um objeto de projeto no sistema DAMASCUS [6], desenvolvido na Universidade de Karlsruhe. Esta hierarquia contém **Representações** (que correspondem a vistas), **Alternativas**, **Revisões** e **Estágios de Projeto**, estes dois últimos correspondendo a versões. Alternativas sob uma mesma Representação são em geral independentes umas das outras, mas o usuário pode estabelecer relações *derivada-de* entre elas. Revisões de uma mesma Alternativa estão organizadas na forma de um grafo acíclico, que implementa a relação *é-revisão-de*. Cada Revisão pode evoluir ao longo do tempo, originando

[4] A Figura 8 foi extraída de [4]

uma seqüência linear de Estágios de Projeto, onde estão de fato armazenados os dados de projeto.

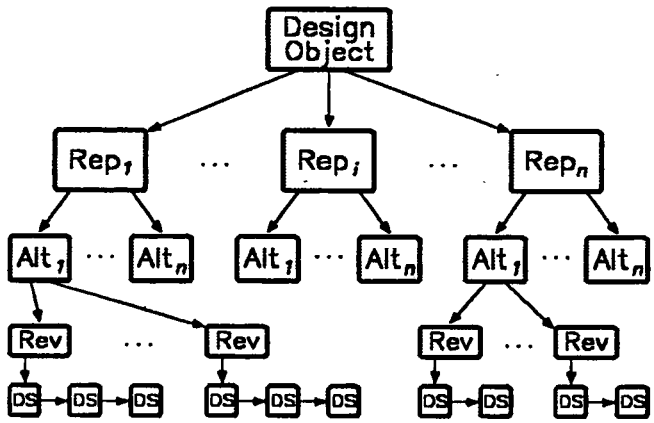


Figura 8: Estrutura de controle do sistema DAMASCUS

O modelo de dados DAMASCUS pode ser mapeado para o GARDEN de uma maneira quase direta, conforme ilustrado na Figura 9. Representações DAMASCUS são mapeadas para ViewGroups que não possuem atributos de interface. Alternativas DAMASCUS de uma Representação R_d são implementadas através de Views GARDEN sob o ViewGroup que corresponde a R_d, e a declaração da interface é armazenada neste nível da estrutura de controle.

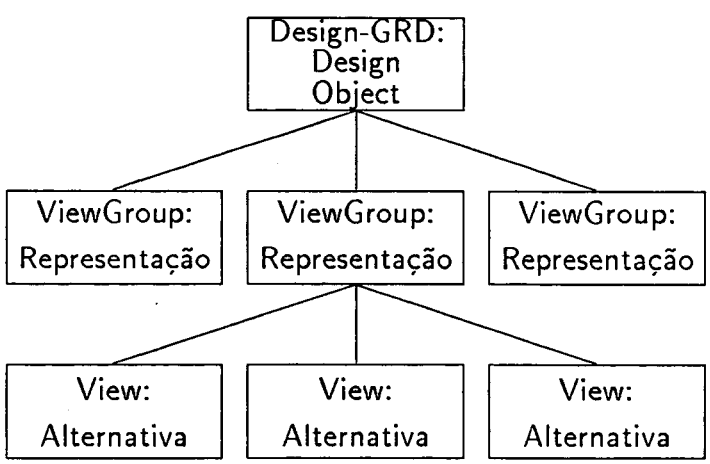


Figura 9: Mapeamento da estrutura de controle DAMASCUS para GARDEN

O mapeamento de Representações para ViewGroups e de Alternativas para Views pode parecer pouco natural, já que uma associação imediata de idéias levaria ao mapeamento de Alternativas para ViewGroups (ambos correspondem a

decisões de projeto) e de Representações para Views (ambos correspondem a níveis de projeto). Deve-se no entanto lembrar a dualidade do conceito de ViewGroup [4], que permite a modelagem não só de uma hierarquia de decisões de projeto como também a associação de níveis de abstração às decisões de projeto. Estritamente do ponto de vista de sua implementação, este mapeamento não apresenta nenhuma dificuldade ou desvantagem.

Relações de equivalência DAMASCUS entre Alternativas de diferentes Representações podem ser facilmente implementadas através de Correlações GARDEN. O mecanismo de configuração GARDEN pode ser usado diretamente para implementar as configurações DAMASCUS.

O mapeamento do modelo DAMASCUS para GARDEN apresenta duas restrições:

- DAMASCUS permite que sinais de interface sejam acrescentados no nível das Revisões, enquanto GARDEN não pode fazê-lo no nível de Modificações e Iterações;
- enquanto DAMASCUS organiza Revisões na forma de um grafo acíclico, GARDEN armazena Modificações e Iterações na forma de uma árvore (que não pode implementar um grafo).

4 Gerência de metodologias de projeto em GARDEN

Nesta seção é apresentado o modelo de gerência de metodologias de projeto proposto neste trabalho. Antes da apresentação, é necessária uma definição dos termos que serão adotados no modelo, em especial "estruturas de controle genéricas" e "exemplares" [5] e "metodologia de projeto". Segue-se então uma descrição dos fundamentos do modelo de gerência. Após são detalhados os aspectos de uma estrutura de controle "genérica" (que como se verá está intimamente associada à idéia de metodologia de projeto) que devem ser especificados para que se construa um ambiente concreto de projeto sobre o GARDEN. É então explicada a relação entre a metodologia de projeto e o processo de integração de ferramentas ao ambiente. Por último é mostrado de que forma a execução de ferramentas vai gerando as estruturas de controle "exemplares" que realmente contêm os dados de um projeto particular.

Esta seção mostra o que deve ser feito para que se defina uma estrutura de controle "genérica" e portanto uma metodologia de projeto. Não é sugerido em nenhum momento **como** isto será feito, ou seja, qual ferramenta será oferecida aos construtores de aplicações para esta definição. A descrição desta ferramenta deve ser objeto de um trabalho posterior.

Deve também ficar claro que, em se tratando de uma proposta preliminar, muitas das facilidades oferecidas não são detalhadas, o que deverá ser feito quando da definição e construção da ferramenta que implementará a gerência de metodologias de projeto.

4.1 Definição de termos

Uma estrutura de controle **genérica** é um *template* que define de que forma pode ser organizada a hierarquia de vistas, alternativas e versões (ver a definição mais rigorosa destes termos em [1]) sob um objeto de projeto [6]. Exemplos de estruturas de controle genéricas suportadas pelos ambientes GARDEN, Oct e DAMASCUS e pela linguagem VHDL foram mostrados nas Figuras 1, 4, 6 e 8, respectivamente.

Uma estrutura de controle **exemplar** é uma instância concreta de uma estrutura genérica, gerada ao longo de um processo de projeto em função da criação de objetos por ferramentas diversas. A Figura 3 mostra parte da estrutura exemplar criada para um objeto *Bloco Operacional* por uma dada metodologia de projeto, seguindo a estrutura de controle genérica do ambiente GARDEN. Esta estrutura exemplar contém vistas com descrições em diferentes níveis de projeto. Em [3] encontra-se

[5] Estas denominações foram inspiradas pela terminologia adotada por S.Wendt no projeto CALVADOS da Universidade de Kaiserslautern para distingüir entre *Datengitter* ("grades de dados") genéricas e exemplares

[6] Cada ambiente tem sua própria terminologia para objetos de projeto, vistas, alternativas e versões

a representação completa desta estrutura exemplar, que contém um número bem maior de ViewGroups, Views e ViewStates.

Uma **metodologia de projeto**, no contexto deste trabalho, é um conjunto de regras que forçam ou orientam as atividades de projeto executadas pelo usuário, de modo a obter objetos de projeto com propriedades desejadas, obedecendo a restrições e atingindo os objetivos de projeto. A estrutura exemplar mostrada na Figura 3 é reflexo direto de uma dada metodologia de projeto. Nesta metodologia, por exemplo, é previsto que haverá a seguinte sucessão de passos para o projeto de um módulo a ser incorporado em um circuito integrado:

- No primeiro passo, é escolhida uma arquitetura para o fluxo de dados do sistema. No exemplo o projetista criou duas alternativas, uma com barramentos e outra com multiplexadores. A metodologia prevê que cada alternativa criada neste nível gerará um ViewGroup diferente sob a raíz do Design. A descrição de cada arquitetura (em termos de registradores, unidades funcionais e interconexões) é armazenada como uma View HDL sob o ViewGroup correspondente.

- No passo seguinte, é decidido qual esquema de relógio será adotado, o que também define os tipos de registradores, se estáticos ou dinâmicos (e neste caso qual o número de fases de relógio). Esta etapa mapeia uma dada arquitetura para a tecnologia escolhida. No exemplo foram criadas alternativas com relógio de 2 e de 4 fases. Estas alternativas são armazenadas como ViewGroups distintos sob o ViewGroup da arquitetura escolhida. A descrição da arquitetura, agora já com os detalhes do esquema de relógio, é armazenada como uma View HDL sob o ViewGroup correspondente.

- Num terceiro passo, é escolhida uma estratégia de implementação (standard-cell, gate-array ou outra). Para cada estratégia a ser avaliada é criado um ViewGroup sob o ViewGroup do esquema de relógio escolhido.

A metodologia de projeto não deve portanto unicamente definir qual a seqüência de ferramentas a ser ativada. Ela deve também prever de que forma será gerada a estrutura de controle exemplar para cada objeto projetado. O detalhamento desta função obviamente depende da estrutura de controle genérica suportada pelo ambiente no qual rodam as ferramentas de projeto. Nas subseções a seguir se verá, no caso do sistema GARDEN, quais regras devem ser especificadas para que o ambiente saiba como gerar uma estrutura de controle exemplar.

Pode-se agora dar uma definição mais completa de metodologia de projeto, tal como entendida no escopo deste trabalho. Uma metodologia de projeto prescreve:

- qual seqüência de ativação de ferramentas deve ser executada, e

- como deve ser gerada a estrutura de controle exemplar para cada objeto projetado por esta metodologia.

A relação entre metodologia de projeto e estrutura de controle genérica também fica clara a partir desta definição, já que uma estrutura de controle exemplar deve necessariamente ser uma instância da estrutura de controle genérica suportada pelo ambiente. Na prática, a definição da metodologia de projeto está limitada pelas possibilidades oferecidas pela estrutura genérica. Além disto, as estruturas genéricas dos diversos ambientes sempre possuem alguma flexibilidade, deixando para o projetista ou o construtor da aplicação a definição de determinados objetos, relações e atributos. Para que se defina como deve ser criada uma estrutura de controle exemplar por meio de uma metodologia de projeto, é necessário portanto que também se personalize a estrutura genérica, definindo estes objetos, relações e atributos. Na subseção 4.3 se verá quais definições devem ser feitas no caso do sistema GARDEN.

Entende-se por **gerência de metodologia de projeto** a função que permite a um ambiente o controle da execução das ferramentas, de tal modo que sejam respeitadas as possíveis seqüências de ativação de ferramentas e que sejam criadas estruturas de controle exemplares válidas, tal como especificado na metodologia de projeto. São então necessários dois mecanismos no ambiente:

- um mecanismo que permita a definição da metodologia de projeto (como já foi alertado, não é objetivo deste relatório descrever a ferramenta que implementa este mecanismo), e
- outro mecanismo que controle a execução das ferramentas de acordo com a metodologia especificada (mecanismos com esta função serão discutidos brevemente na seção 6, não sendo também objetivo deste relatório o seu detalhamento).

A gerência de metodologias de projeto é uma das funções principais de um *design framework*. Distingüe-se aqui claramente entre os conceitos de **ambiente** e **framework**:

- Um *framework* é um conjunto de recursos para a construção de ambientes de projeto; entre estes recursos estão o suporte a uma estrutura de controle genérica e um mecanismo de gerência de metodologias de projeto.
- Um ambiente é um conjunto particular de ferramentas, integradas pelos recursos oferecidos por um *framework*, de tal modo a implementar uma dada aplicação. Um ambiente pode ser restrito a uma dada arquitetura de sistemas, ou a uma dada tecnologia de fabricação, ou a uma dada metodologia de projeto, ou pode ser flexível a respeito de um ou mais destes aspectos.

É necessário aqui que se distingua entre os vários tipos de usuários envolvidos com um *framework*. Numa primeira abordagem, pode-se pensar em apenas dois usuários extremos e distintos:

- O **construtor da aplicação** é o usuário que utiliza os recursos do *framework* para definir a metodologia de projeto a ser seguida e promove a integração de ferramentas, de modo a criar um ambiente de projeto.

- O **projetista** é o usuário que projeta circuitos e sistemas utilizando a metodologia e as ferramentas colocadas à sua disposição.

Pode-se no entanto pensar em usuários com gradações variadas entre estes dois extremos. Um ambiente pode oferecer aos projetistas flexibilidade para a definição de metodologias de projeto particulares, seja para determinados projetos (a metodologia deve então ser seguida por um grupo de projetistas) ou para projetistas isolados.

A **integração de ferramentas** é uma das tarefas a serem executadas pelo construtor da aplicação. As ferramentas devem ser integradas de uma maneira consistente com a metodologia de projeto, e em particular com a estrutura de controle imposta aos projetistas.

Falando de forma genérica, há dois graus de integração possíveis em ambientes de projeto, a forte e a fraca [1] (também identificados por *white-box* e *black-box* [8] ou por "integração de ferramentas" e "encapsulamento de ferramentas" [9]).

Na integração fraca, o código da ferramenta não é alterado, o que é conveniente para uma rápida integração de ferramentas já existentes. É montada, à volta da ferramenta, uma descrição que mapeia os objetos manipulados pela ferramenta em objetos da estrutura de controle. Este tipo de integração é normalmente adotado em conjunto com estruturas de controle de "larga granularidade" [1], isto é, onde os objetos (Designs, em GARDEN) mais primitivos para os quais são construídas estruturas de controle exemplares são "células" ou "módulos" sem semântica pré-definida (p.ex. "Bloco Operacional", "Banco de Registradores", "Unidade Aritmética" e assim por diante).

Na integração forte, o código da ferramenta é re-escrito de modo que todo acesso a um dado de projeto é mapeado para um acesso a um objeto da estrutura de controle suportada pelo ambiente (ou então a ferramenta é originalmente desenvolvida para o ambiente em particular, já com conhecimento da estrutura de controle). Este tipo de integração pode então mais facilmente ser adotado em conjunto com estruturas de controle com "granularidade fina", onde os objetos primitivos para os quais são construídas estruturas exemplares são "portas lógicas", "registradores", "transistores", etc.

Pode-se obviamente adotar uma integração fraca também para estruturas com granularidade fina. A eficiência desta solução pode no entanto ficar bastante comprometida. Imagine-se por exemplo a grande quantidade de acessos à base de dados e mapeamentos que deveriam ser feitos antes da execução de um DRC (verificador de regras de projeto), para preparar os dados necessários à verificação de *layouts* de circuitos de grande complexidade.

Em qualquer caso, é evidente que o processo de integração de uma ferramenta deve ser feito de forma consistente com a estrutura de controle genérica imposta pela metodologia de projeto. A definição da estrutura de controle genérica portanto precede o processo de integração de ferramentas.

4.2 Fundamentos do modelo de gerência

A Figura 1 mostra a estrutura de controle genérica suportada por GARDEN. A Figura 3 mostra uma possível estrutura de controle exemplar construída com base naquela estrutura genérica. GARDEN apresenta duas propriedades não comumente encontradas em outros modelos de representação e gerenciamento de dados:

- ele oferece uma estrutura genérica com uma hierarquia de ViewGroups de profundidade variável, e
- ViewGroups são agrupadores de representações que têm uma semântica aberta (ou seja, pode-se adotar diferentes critérios de agrupamento de ViewGroups e Views sob um ViewGroup).

É esta flexibilidade que permite justamente o mapeamento de diferentes estruturas de controle genéricas para a estrutura genérica do GARDEN, tal como sugerido na seção 3. As Figuras 5, 7 e 9 mostram estruturas que ainda são genéricas, mas que são implementadas sobre a estrutura genérica da Figura 1. Tomemos como exemplo o mapeamento da estrutura de controle de DAMASCUS para o GARDEN. Um ambiente implementado sobre a estrutura da Figura 9 manipula ao menos três níveis de estrutura de controle:

- no primeiro nível está a estrutura genérica suportada por GARDEN;
- no segundo nível está a estrutura genérica DAMASCUS, implementada na forma de um mapeamento para a estrutura genérica GARDEN; e
- no terceiro nível estão as estruturas exemplares que podem ser construídas a partir de uma metodologia que implemente a estrutura genérica do segundo nível.

Neste ambiente, a estrutura exemplar mostrada na Figura 3 não poderia ser gerada, já que ela supõe uma flexibilidade (por exemplo número variável de níveis de ViewGroup na hierarquia da estrutura de controle) não presente na estrutura genérica de segundo nível.

Estas considerações definem a base do método de gerência de metodologias de projeto proposto neste trabalho. Ele fundamenta-se em dois princípios:

1. *Sobre a estrutura de controle genérica GARDEN, de agora em diante denominada estrutura genérica* **primária**, *pode ser definida uma hierarquia de estruturas genéricas* **secundárias**, *cada qual herdando todas as propriedades da estrutura do nível anterior.*
2. *Um projetista pode situar-se em qualquer nível desta hierarquia de estruturas genéricas, sendo a ele imposta a metodologia de projeto definida no nível onde ele está situado.*

Toda estrutura de controle genérica secundária acrescenta semântica à estrutura primária (ou a uma seqüência "estrutura primária ⇒ estrutura secundária ⇒ ... ⇒

estrutura secundária"). Esta semântica adicional é obtida restringindo-se a flexibilidade oferecida no nível anterior da hierarquia de estruturas genéricas, por exemplo definido-se uma topologia básica de ViewGroups e critérios de agrupamento de Views sob estes ViewGroups ou criando-se atributos e relações particulares. Como conseqüência deste princípio e das considerações feitas na subseção anterior, estas restrições sucessivas sobre a estrutura de controle significam metodologias de projeto cada vez mais detalhadas e restritivas.

A Figura 10 ilustra este princípio. As estruturas genéricas $ES_1, ..., ES_n$ são definidas sobre a estrutura genérica primária EP, herdando suas propriedades e restringindo sua flexibilidade. As estruturas genéricas $ES_{21}, ..., ES_{2m}$, por sua vez, fazem o mesmo sobre ES_2.

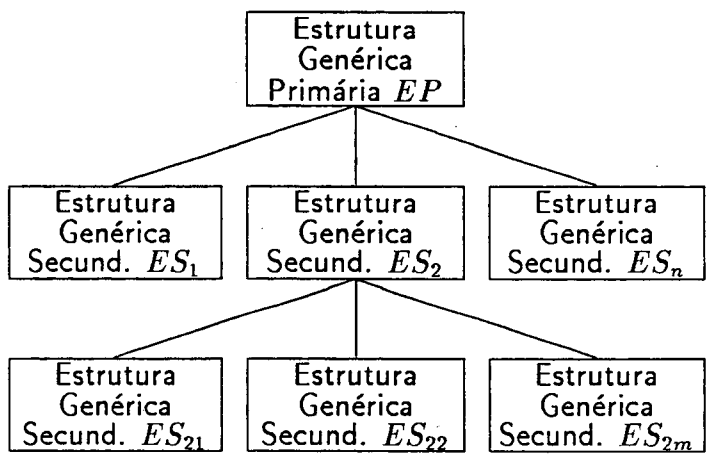


Figura 10: Hierarquia de estruturas de controle genéricas

Importante neste modelo de gerência de metodologias de projeto é a flexibilidade oferecida na construção de um ambiente. O *framework* não impõe uma certa hierarquia de usuários, do tipo "construtor da aplicação ⇒ projetista". Pode-se, ao contrário, definir diferentes níveis de usuários, que têm graus de liberdade variáveis na definição de metodologias de projeto. Assim, todo usuário é associado a um certo nível de uma hierarquia de estruturas de controle genéricas. Ele pode seguir a metodologia de projeto imposta neste nível, ou definir uma metodologia ainda mais restritiva (i.e. detalhada) no caso de um projeto particular.

Este modelo deixa ainda aberta a possibilidade de se definir uma instalação com um número fixo de níveis de estrutura de controle genérica, com uma alocação pré-definida de "tipos de usuários" a cada nível (por exemplo "administrador geral", "administrador de projeto" e "projetista"), cada qual com atribuições específicas quanto à definição de estruturas de controle – i.e. metodologias de de projeto – e à integração de ferramentas. Os projetistas poderiam por exemplo estar limitados a seguir uma metodologia de projeto imposta pela hierarquia de estruturas, sem liberdade de definição de metodologias particulares.

No processo de integração de ferramentas, é especificado para cada ferramenta de que modo ela está relacionada à estrutura de controle genérica a ela imposta pela metodologia de projeto (estrutura esta resultante de uma hierarquia de estruturas secundárias). Nesta especificação consta de que forma os dados manipulados pela ferramenta são mapeados para os objetos da estrutura de controle. Toda flexibilidade eventualmente existente na estrutura de controle é então repassada ao projetista.

Explicados os fundamentos do modelo de gerência de metodologias, pode-se agora compreender porque é preferível não se adotar a terminologia típica de banco de dados para designar os "modelos de dados" suportados em GARDEN. Pode-se imaginar a estrutura de controle genérica primária como sendo um "modelo de dados semântico", orientado à aplicação de ambientes de projeto. Sobre este "modelo" podem ser definidos de maneira hierárquica "esquemas conceituais" para aplicações sucessivamente mais restritivas e detalhadas. Como este conceito não é comparável aos recursos tradicionalmente oferecidos em sistemas de gerenciamento de banco de dados, é preferível a adoção de uma terminologia distinta e particular.

4.3 Definição de uma estrutura de controle genérica secundária

Nesta subseção serão detalhadas as definições a serem feitas para que se especifique uma estrutura de controle genérica secundária a partir da estrutura genérica primária do GARDEN, mostrada nas Figuras 1 e 2 e explicada na seção 2. A estrutura genérica primária é bastante flexível e não possui características particulares de nenhuma aplicação. Cada definição numa estrutura secundária pode

- definir a topologia básica da árvore de ViewGroups e Views;
- estender a estrutura de controle para uma dada aplicação, especificando atributos, correlações, tipos de Views e objetos auxiliares; e
- restringir a estrutura de controle, por exemplo definindo como serão agrupados objetos sob ViewGroups, ou quais referências são possíveis em configurações, ou ainda especificando restrições de integridade não previstas na estrutura genérica primária.

4.3.1 Definição da topologia básica da estrutura de controle

Nas Figuras 7 e 9 foram mostradas estruturas de controle com diferentes critérios de agrupamento de ViewGroups e Views sob os ViewGroups:

- Na Figura 7, os ViewGroups diretamente sob a raíz da estrutura de controle (o Design) tem cada um deles unicamente um ViewGroup sob si (no qual está definida a interface da Contents Facet de Oct). Sob este ViewGroup são agrupados uma View na qual é definido o conteúdo real da Contents Facet e

um número qualquer de Views que armazenam Interface Facets. Estas Views podem ser de tipo MMHD ou Layout, dependendo da Faceta.

- Na Figura 9, sob o Design existem ViewGroups que correspondem a diferentes Representações DAMASCUS (cada um destes ViewGroups deve ter um atributo que define um nível de abstração). Sob cada um destes ViewGroups há um número qualquer de Views, correspondendo a diferentes alternativas de projeto.

Na metodologia de projeto deve portanto ser definida a topologia genérica da estrutura de controle. Devido ao mecanismo de herança de propriedades de uma estrutura de controle para as estruturas dela derivadas (suas estruturas descendentes na hierarquia), a definição de estruturas de uma forma hierarquizada deve obedecer às seguintes regras:

- Um ViewGroup ou uma View definido numa estrutura de controle secundária não pode ser removido por uma estrutura secundária dela derivada.
- Uma estrutura secundária pode acrescentar livremente ViewGroups e Views sob os nodos da estrutura genérica criada pela hierarquia de onde ela é derivada.

A regra de formação da topologia pode no entanto restringir as estruturas de controle possíveis de serem construídas, prevendo, para o Design e para cada ViewGroup:

- quantos ViewGroups e Views podem ser armazenados (eventualmente só ViewGroups ou só Views) sob o ViewGroup ou Design;
- regra de formação do nome destes ViewGroups e Views; e
- restrições de integridade que devem ser respeitadas pelas Views e/ou ViewGroups a serem colocados sob o ViewGroup ou Design (uma restrição simples poderia ser relacionada ao valor de um atributo).

No processo de integração de ferramentas deve-se especificar, de forma consistente com a estrutura de controle imposta pela metodologia ao projetista, em que ponto da estrutura deve ser colocado cada objeto criado por cada ferramenta (ver subseção 4.4).

GARDEN prevê o armazenamento na estrutura de controle, junto a cada ViewGroup, de um atributo especial (um *string*) denominado "critério de agrupamento". O valor deste atributo tem um efeito meramente documentacional. No caso do critério de agrupamento ter sido fixado na metodologia de projeto, o valor deste atributo deve ser atribuído pelas ferramentas de projeto. Em caso de um agrupamento decidido livremente pelo projetista, este deve atribuir o valor ao atributo (a interface do ambiente com o projetista deve permitir esta facilidade).

Sob cada View podem ainda ser armazenados inúmeros ViewStates (Modificações e Iterações), na forma de uma árvore. O critério para armazenamento

de um novo ViewState como Modificação ou Iteração de algum ViewState anterior deve ser deixado inteiramente a cargo do projetista, já que a diferença entre estes conceitos é subjetiva. Uma metodologia de projeto particular pode no entanto restringir esta árvore, p.ex. permitindo apenas o armazenamento de uma seqüência linear de Iterações, não suportando Modificações.

4.3.2 Outras definições de extensão

Atributos GARDEN permite que sejam definidos atributos em virtualmente todos os objetos da estrutura de controle: Repositório, Processo, Biblioteca, Design, ViewGroup, View, Componente, Conexão (estes dois últimos compõem uma View MMHD), Porta, Instância de Porta e Correlação. Um atributo é um *string* ou um arquivo cujo conteúdo é interpretado pelas ferramentas de projeto.

Pode-se pensar em duas classes de atributos:

- atributos definidos pelo projetista, associados por ele a objetos quaisquer, com fins documentacionais particulares, e que não afetam a metodologia de projeto; e
- atributos definidos pela aplicação, e portanto especificados na hierarquia de estruturas de controle.

As ferramentas de projeto devem suportar a definição de tipos de atributos (i.e. definição do nome do atributo e do seu intervalo de valores) da primeira classe, assim como a criação de instâncias destes tipos, atribuição de valores a eles e recuperação de seus valores.

No escopo deste trabalho interessam em particular os atributos da segunda classe. Um atributo de um objeto qualquer (um ViewGroup, por exemplo), por ser definido na metodologia de projeto, existirá para todas as instâncias deste objeto criadas pelo projetista. Deve-se poder distingüir entre atributos obrigatórios e opcionais, e para todos eles deve ser possível a atribuição de um valor *default* na própria definição da metodologia de projeto.

Exemplos de atributos são:

- Área ocupada, freqüência máxima de operação e potência consumida por um módulo (atributos atribuíveis a Design, ViewGroup, View ou ViewState, dependendo da metodologia de projeto) – estes atributos são utilizados como restrições a serem obedecidas por programas de síntese, podendo sua verificação ser utilizada para controlar seqüências de ativações de ferramentas.
- Dimensões de uma interface e fatores de escala máximos que podem ser aplicados sobre estas dimensões (atributos atribuíveis a Design, ViewGroup e/ou View, dependendo do nível da estrutura de controle no qual for armazenada a definição da interface do objeto) – estes atributos são utilizados por ferramentas de composição automática de *layout*.

- Atributos geométricos de Portas, tais como localização ao longo da envoltória de uma célula, largura e camada do *layout* em que estão implementadas – estes atributos são utilizados por ferramentas de roteamento.
- Atributos geométricos de Conexões, tais como traçado (seqüência de segmentos de reta) e camada do *layout* em que estão implementadas (eventualmente cada segmento de reta pode estar implementado numa camada distinta) – estes atributos são gerados por ferramentas de roteamento e posteriormente analisados por ferramentas de extração de parâmetros elétricos para a avaliação exata de *timing*.

Na metodologia de projeto deve-se definir, para um dado tipo de objeto (ou para um tipo de objeto com um certo valor de algum outro atributo), que tipos de atributos ele tem (nome, se *string* ou arquivo e valor *default*). Valores concretos serão atribuídos a atributos de instâncias de objetos deste tipo. A atribuição pode ser feita manualmente, pelo projetista, ou automaticamente, por uma ferramenta de projeto (neste caso a atribuição deve ser prevista no processo de integração da ferramenta).

Atributos podem ser usados em particular para definir níveis de projeto. Por sua importância no controle de metodologias de projeto, este uso será analisado de forma separada no item a seguir.

Tipos de Views Conforme Gajski e Kuhn [10], todo projeto é feito em níveis situados ao longo de três eixos de projeto: comportamental, estrutural e geométrico. Estes eixos são mapeados para três tipos de Views em GARDEN, respectivamente HDL, MMHD e Layout. As Views de GARDEN correspondem portanto a "níveis genéricos" sobre estes eixos, não possuindo uma semântica própria que defina algum nível particular sobre eles. Esta particularização pode ser feita apenas por atributos associados às Views. Assim, pode-se definir um atributo de nome *nível de abstração* para cada View, de tipo *string*, que por exemplo pode assumir valor

- "algoritmo", "rede de Petri" ou "máquina de estados", para Views HDL (ou alternativamente "VHDL", "ISP" ou "KARL", caso se associe nomes de linguagens de descrição de hardware aos níveis);
- "RT", "lógico" ou "elétrico" para Views MMHD;
- "planta baixa", "máscara simbólica" ou "máscara" para Views Layout.

Correlações Correlações podem ser estabelecidas entre dois ou mais objetos quaisquer (por exemplo entre uma Biblioteca e um Design, ou entre um ViewGroup e uma View, ou entre dois ViewStates). Correlações servem ao armazenamento de relações quaisquer entre os objetos (ver [1]). Uma correlação de grande importância é a relação de equivalência entre dois ou mais objetos, significando que eles foram gerados por síntese de uma mesma descrição comum ou que, através de simulação

ou de verificação formal, o projetista estabeleceu a equivalência entre eles. Relações de equivalência podem ser estabelecidas por exemplo:

- entre dois ou mais ViewStates de um mesmo Design, ou de um mesmo ViewGroup, ou de uma mesma View;
- duas ou mais Views de um mesmo Design ou ViewGroup, e portanto entre todos os ViewStates destas Views;
- entre uma View (portanto todos os ViewStates desta View) e um ViewGroup (portanto todas as Views e ViewStates deste ViewGroup).

A metodologia de projeto deve prever o local da estrutura de controle onde será armazenada uma correlação e quais os tipos de objetos que podem ser correlacionados. Da mesma forma que para os atributos, os nomes dos objetos concretos correlacionados (portanto o "valor" da correlação) serão atribuídos pelo projetista ou por uma ferramenta de projeto.

Estruturas de controle alternativas Uma metodologia de projeto deve ter liberdade de prever diferentes estruturas de controle para diferentes objetos. Na Figura 3, por exemplo, é mostrada uma estrutura de controle exemplar que teria sido construída a partir da definição de uma estrutura genérica apropriada a um objeto "Bloco Operacional". Em [3] pode-se ver que a estrutura exemplar de um objeto "Bloco de Controle" é diferente e exigiria outra definição para a a estrutura genérica. Numa hierarquia de estruturas secundárias, estas duas estruturas genéricas (para objetos "Bloco Operacional" e para objetos "Bloco de Controle") poderiam ser derivadas de uma mesma estrutura genérica secundária. A estrutura de controle a ser seguida quando da criação de um novo objeto poderia ser fixada previamente pela metodologia de projeto, ou o usuário poderia ter liberdade de definição da estrutura de controle à medida que criasse representações para o objeto em questão. No primeiro caso, a estrutura de controle seria conhecida pela ferramenta que cria as representações do objeto (por exemplo um gerador de partes operativas seria necessariamente limitado a criar objetos do tipo "Bloco Operacional", como na Figura 3).

Note-se que é também possível definir-se estruturas de controle alternativas a partir de metodologias de projeto alternativas. A partir de uma metodologia M_1 podem ser derivadas duas metodologias M_2 e M_3. No exemplo antes mencionado, M_2 seria orientada a objetos "Bloco Operacional", enquanto M_3 seria orientada a objetos "Bloco de Controle". Ferramentas utilizadas na geração de blocos operacionais seriam integradas em M_2, enquanto ferramentas para a geração de blocos de controle o seriam em M_3. M_1 conteria uma definição de estrutura de controle genérica comum aos dois tipos de blocos. Esta solução será exemplificada com maiores detalhes na subseção 5.3.

Objetos auxiliares GARDEN não prevê a definição de objetos auxiliares, tais como estímulos de simulação e informações de testabilidade (ver [1]). Estes objetos só podem ser criados através de uma extensão à estrutura de controle. A ferramenta de definição da metodologia de projeto deve portanto permitir que se especifique, para um objeto auxiliar,

- seu nome,
- eventualmente seus atributos, da mesma forma que são definidos os atributos dos demais objetos (um atributo interessante é o nome da ferramenta que criou este objeto auxiliar), e
- eventualmente correlações com outros objetos, da mesma forma como são definidas as demais correlações.

O conteúdo de um objeto auxiliar será um arquivo a ser interpretado unicamente pelas ferramentas de projeto.

4.3.3 Outras definições de restrição

Nenhuma das definições a seguir é obrigatória. Caso não haja, numa estrutura genérica secundária, uma definição que restrinja a estrutura num certo aspecto, então a metodologia de projeto imposta aos projetistas mantém a flexibilidade da estrutura genérica primária. Esta flexibilidade pode então ser aproveitada no processo de integração de ferramentas, sendo repassada aos projetistas, como será visto na subseção 4.5. Uma primeira restrição, bastante importante e já mencionada na subseção 4.3.1, é a definição de critérios de agrupamento de ViewGroups e Views sob os ViewGroups.

Definição da interface A estrutura genérica primária de GARDEN permite que sinais de interface ("Ports") sejam armazenados em qualquer nível da estrutura, desde o Design até as Views. A interface completa de uma View é formada por um mecanismo de herança desde o Design. Uma metodologia de projeto particular pode no entanto prever o armazenamento dos Ports apenas em determinados pontos da estrutura de controle. Assim, por exemplo:

- Na Figura 5 os Ports são todos armazenados no nível do Design.
- Na Figura 7 os Ports são todos armazenados no ViewGroup que mapeia a Contents Facet. As Views sob este ViewGroup que mapeiam as Interface Facets podem acrescentar determinados atributos à interface ou a seus Ports, mas não podem acrescentar Ports.
- Na Figura 9 os Ports são todos armazenados no nível das Views (na realidade DAMASCUS permite que Ports sejam acrescentados no nível de Revisões, que correspondem aos ViewStates, mas isto não é possível em GARDEN).

Uma estrutura de controle genérica secundária deve poder então restringir os níveis nos quais Ports são armazenáveis. Propõe-se combinar esta restrição com o mecanismo de herança de propriedades entre estruturas secundárias, de modo que uma metodologia de projeto possa prever três atributos para o armazenamento de Ports num nível qualquer da estrutura de controle:

- armazenamento proibido – nenhuma estrutura secundária derivada desta pode permitir o armazenamento de Ports neste nível;
- armazenamento previsto – toda estrutura secundária derivada desta deve permitir o armazenamento de Ports neste nível; e
- armazenamento não previsto – uma estrutura secundária derivada desta pode estabelecer o atributo "previsto", "não previsto" ou "proibido" para o armazenamento de Ports neste nível.

Configurações GARDEN permite cinco tipos diferentes de referências a objetos em um componente (uma instância de outro objeto):

- referência a um Design,
- referência a um ViewGroup de um Design,
- referência a uma View de um Design,
- referência explícita a um ViewState de um Design, e
- referência genérica ao ViewState mais recente de um Design.

Exceto pelos dois últimos casos, nos quais a escolha do ViewState já está definida na descrição do objeto que contém a instância, nos demais é necessária a seleção posterior de um ViewState particular (o que pode exigir também a seleção de uma View e/ou de ViewGroups) para que a descrição que contém este componente possa ser processada por alguma ferramenta de projeto.

Uma metodologia de projeto pode especificar quais tipos de referências (dentre as cinco citadas acima) são permitidas de forma estática numa descrição. Pode-se, por exemplo, restringir a referência a Designs que estão na mesma Biblioteca que contém o Design cujo ViewState faz a referência.

Uma metodologia de projeto pode também especificar restrições relativas à escolha de ViewGroups, Views e ViewStates numa configuração dinâmica. Como exemplo pode-se restringir a seleção a ViewStates de Designs de uma Biblioteca B_i cujo Processo P_i esteja incluído no Processo P_j da Biblioteca B_j que contém a descrição onde está a referência a ser configurada.

Existe uma outra função na seleção de objetos para configurações dinâmicas que não deve ser considerada parte da gerência de metodologia de projeto. O projetista pode desejar criar configurações de acordo com restrições relativas a valores de atributos quaisquer dos objetos. Estas restrições devem poder ser variadas livremente pelo projetista enquanto este testa diferentes soluções para o projeto de

determinado módulo ou sistema, motivo pelo qual elas não podem ser fixadas na metodologia de projeto. Exemplos destas restrições são:

- no caso de uma configuração para a composição do *layout* de um CI, selecionar apenas ViewStates (de Views Layout, obviamente) que atendam determinadas restrições de área ("área" seria um atributo de Views Layout);
- no caso de uma configuração para o projeto lógico, selecionar apenas ViewStates com atraso máximo de x ns ("atraso" seria um atributo de Views MMHD que têm um atributo "ViewType" com valor "lógico").

O controle destas restrições deve ser exercido dentro da ferramenta de projeto que vai necessitar da descrição que contém a configuração dinâmica. Pode-se implementar uma ferramenta de configuração especial, ativada pelas diferentes ferramentas de projeto, como proposto por exemplo em [11].

Associação de processos Processos podem ser associados em GARDEN ao Repositório, Bibliotecas, Designs, ViewGroups e Views. Feita uma associação de um processo a um objeto qualquer desta hierarquia, ele é automaticamente imposto a todos os objetos abaixo dele (até o nivel de ViewState, portanto).

A metodologia de projeto deve especificar:

- quais processos existem (seus nomes e o nome dos arquivos que contêm as informações tecnológicas);
- os nomes dos processos incluíveis num processo P_i (como visto no item anterior, processos incluídos podem ser utilizados no controle do processo de configuração); e
- a associação dos diversos processos à hierarquia de objetos, respeitada a restrição de que não se pode associar um processo a um objeto se já há uma associação de outro processo a um objeto acima na hierarquia.

Correlações e atributos, já vistos em itens anteriores, também podem ser definidos para os processos.

Uma estrutura de controle genérica secundária impõe a todas as estruturas secundárias dela derivadas a associação de um processo a um dado nivel da hierarquia de objetos. A associação não pode ser especificada se já há uma associação numa estrutura secundária superior na hierarquia.

Restrições de integridade Dentre os vários tipos de restrições de integridade que devem ser verificadas num ambiente de projeto [1], algumas são relativas à gerência de metodologia de projeto. Exemplos de restrições que podem ser especificadas numa dada metodologia são:

- uma relação de equivalência só deve ser estabelecida entre Views de um mesmo ViewGroup;

- um ViewState ainda não validado (por simulação ou outro método qualquer) não pode ser referenciado em outro objeto;
- uma determinada ferramenta de projeto não pode ser aplicada sobre um certo ViewState se determinados atributos deste ViewState não tiverem seus valores dentro de certos limites.

Não há um limite aos tipos de restrições de integridade que podem ser especificadas numa metodologia de projeto. É desejável portanto que o construtor da aplicação tenha à sua disposição uma linguagem (eventualmente a mesma utilizada na especificação da metodologia de projeto) que permita descrever restrições envolvendo ferramentas, objetos e seus atributos. Não é objetivo deste trabalho descrever tal linguagem nem o mecanismo que implementa a verificação das restrições durante a execução das tarefas de projeto.

4.4 Processo de integração de ferramentas

No processo de integração de uma ferramenta, é especificado de que modo ela está relacionada à estrutura de controle genérica a ela imposta pela metodologia de projeto (estrutura esta resultante de uma hierarquia de estruturas secundárias). Este relacionamento se dá através de um mapeamento entre os dados manipulados pela ferramenta e os objetos da estrutura de controle.

GARDEN não faz nenhuma restrição ao grau de integração das ferramentas, o que equivale a dizer que ele não restringe a granularidade desejada para a estrutura de controle. Objetos GARDEN são criados e recuperados através de uma interface procedural de dados [2]. As funções definidas nesta interface podem ser executadas tanto diretamente pelas ferramentas de projeto, no caso de uma integração forte em que o código da ferramenta é desenvolvido ou alterado de forma consistente com a estrutura de controle genérica, como por uma ferramenta de encapsulamento, adotada no caso de uma integração fraca, que executa dois grupos de ações:

- antes da execução da ferramenta de projeto, busca na base de dados os objetos GARDEN que serão necessários na tarefa, mapeando-os para os dados da ferramenta;
- após a execução da ferramenta de projeto, mapeia os dados criados por esta em objetos GARDEN e armazena estes na base de dados.

Estes dois graus de integração representam portanto na prática dois processos bastantes distintos de integração de ferramentas:

- na integração forte, o mapeamento entre os dados da ferramenta e a estrutura de controle está implícito no código da ferramenta;
- na integração fraca, o mapeamento está definido de forma externa à ferramenta, sendo realizado antes e/ou depois desta ser executada.

Se por um lado a integração forte tem vantagens no maior controle que o ambiente pode exercer automaticamente sobre a consistência do projeto, a integração fraca permite uma flexibilidade muito maior na adaptação das ferramentas a diferentes estruturas de controle. Esta flexibilidade é particularmente útil em GARDEN, onde se pode definir uma hierarquia de estruturas de controle genéricas e associar um projetista a diferentes níveis desta hierarquia.

A integração fraca acima descrita tem no entanto uma grande desvantagem na execução de ferramentas que interagem de forma bastante freqüente com a base de dados. Exemplos são ferramentas interativas como editores de esquemáticos, que criam e buscam objetos na base de dados durante sua execução. É inviável buscar previamente os objetos, já que o projetista poderá desejar objetos quaisquer. Além disto, objetos criados durante a edição devem ser imediatamente armazenados na base de dados, já que deve ser testada sua consistência contra os demais objetos existentes e eles podem ser re-utilizados imediatamente na edição. Como solução para este problema, propõe-se duas variações da integração fraca:

- integração fraca por **mapeamento seqüencial**, na qual o mapeamento é executado antes ou depois da execução da ferramenta de projeto, conforme já descrito;
- integração fraca por **mapeamento concorrente**, na qual o mapeamento é executado durante a execução da ferramenta de projeto, à medida que são feitos acessos aos dados.

Esta segunda variação exige um mecanismo no qual uma camada intermediária que realiza o mapeamento é construída entre a ferramenta e a interface procedural de acesso à base de dados. Esta solução é obviamente mais ineficiente do que a integração forte, na qual a camada intermediária entre a ferramenta e a base de dados não existe.

Como conseqüência da existência de diversas metodologias de projeto possíveis num ambiente, em função dos diversos níveis da hierarquia de estruturas genéricas nos quais um projetista pode atuar, e ainda da possibilidade de se oferecer estruturas de controle alternativas para objetos distintos numa mesma metodologia de projeto, conforme sugerido na subseção 4.3.3, não há uma única especificação possível para a integração de uma ferramenta. O *framework* deve prover uma facilidade para que sejam especificados diferentes mapeamentos de integração para uma dada ferramenta, de acordo com a tarefa a ser executada. Durante a execução de uma seqüência de ferramentas por um dado projetista, o ambiente deve selecionar o mapeamento correto para a invocação da ferramenta, em função da metodologia sendo seguida e da tarefa a ser executada. Esta facilidade tem conseqüências bastante diversas para os diferentes graus de integração:

- No caso de integração forte, haverá vários códigos possíveis para cada ferramenta, cada um deles correspondendo a uma diferente metodologia de projeto.

O ambiente deve selecionar dinamicamente o código correto a ser invocado em função da metodologia / tarefa corrente.

- No caso de integração fraca, haverá diferentes descrições de mapeamento entre os dados da ferramenta e os objetos GARDEN, e o ambiente deve selecionar o mapeamento a ser seguido em função da metodologia / tarefa corrente.

Este segundo caso vale tanto para o mapeamento seqüencial como para o mapeamento concorrente.

No caso da integração fraca, o *framework* deve oferecer uma linguagem para a especificação da integração de ferramentas, tanto no caso do mapeamento seqüencial como concorrente. Esta especificação deve ser consistente com a estrutura de controle genérica imposta aos projetistas. Uma maneira de se facilitar esta verificação de consistência é através de uma linguagem única que sirva simultaneamente aos propósitos de definição da metodologia de projeto e de integração de ferramentas. A linguagem para especificação da integração de ferramentas pode ainda ser a mesma adotada para a descrição das possíveis seqüências de ativação de ferramenta. Como já alertado anteriormente, não é objetivo deste relatório descrever tais linguagens.

4.5 Construção de estruturas de controle exemplares

Em princípio, a definição de uma estrutura de controle genérica (através de uma hierarquia de estruturas genéricas secundárias ou não) cria uma especificação não ambígua da regra de construção das estruturas de controle exemplares. Assim, a cada objeto criado por uma ferramenta de projeto, a metodologia de projeto, em conjunto com a descrição de integração da ferramenta (esta última apenas no caso de integração fraca) decide exatamente onde novos objetos, atributos e relações devem ser acrescentados à estrutura exemplar.

A definição de uma estrutura de controle genérica secundária pode ainda permitir no entanto uma certa flexibilidade. Pode-se por exemplo deixar em aberto os tipos de Views, ou permitir uma hierarquia livre de ViewGroups, ou ainda não impor restrições quanto ao agrupamento de novos ViewGroups e Views sob os já existentes. A metodologia de projeto imposta por uma estrutura de controle com tal flexibilidade deixa portanto decisões para o projetista.

Esta flexibilidade precisa se refletir na interface entre o ambiente e o projetista, e eventualmente na própria implementação das ferramentas. Se por exemplo o agrupamento de objetos sob ViewGroups é livre, então na criação de uma nova View por um editor de esquemáticos o projetista deverá especificar o local exato da estrutura de controle exemplar onde ela deverá ser armazenada. É desejável portanto que os recursos existentes nas linguagens (ou linguagem única) de definição de metodologia de projeto e de integração de ferramentas para identificar objetos, atributos e relações GARDEN possam ser utilizados pelo projetista, de modo que ele possa controlar a construção das estruturas de controle exemplares.

Obviamente a possibilidade acima depende do ambiente que se quer construir. O construtor da aplicação pode desejar que o projetista seja obrigado a trabalhar apenas sobre estruturas genéricas completamente pré-definidas. Esta limitação será no entanto uma propriedade de um ambiente particular construído sobre o *framework*, e portanto este deve oferecer os recursos antes mencionados.

De qualquer forma, deve-se permitir que o projetista crie atributos e correlações (e eventualmente também alguns objetos auxiliares e restrições de integridade) particulares, sem que isto caracterize necessariamente uma nova estrutura de controle genérica secundária e portanto uma nova metodologia de projeto. Também neste caso será necessário oferecer ao projetista meios de referenciar e manipular entidades da estrutura de controle.

Este acesso direto do projetista às estruturas de controle exemplares, tanto para manipulação das mesmas como para simples consultas, tem impacto considerável sobre a interface entre o ambiente e o projetista. *Browsers* têm sido propostos, como em [12], para navegação e consulta em objetos na base de dados de projeto. No caso do GARDEN, no entanto, no qual a estrutura de controle genérica imposta ao projetista é dependente da metodologia de projeto e portanto variável, seria altamente desejável que o *browser* fosse configurado automaticamente pelo ambiente de acordo com a estrutura de controle imposta pela metodologia de projeto corrente.

5 Exemplos de definição de metodologias de projeto

Os exemplos a seguir objetivam ilustrar os princípios de definição de metodologias de projeto e construção de estruturas de controle genéricas apresentados na seção anterior. Eles não pretendem descrever de maneira completa e detalhada metodologias aplicáveis a projetos reais.

5.1 Síntese de alto nível

Este primeiro exemplo apresenta uma seqüência de definições de metodologias de projeto que se "somam", isto é, que acrescentam novas ferramentas e novos objetos à metodologia anterior, sem necessariamente especializá-la ou restringi-la. Após a apresentação da estrutura de controle gerada por cada metodologia, serão dados exemplos de outras definições de extensão e restrição que poderiam ser acrescentadas em cada uma delas.

Metodologia básica A metodologia inicial prevê a síntese, a partir de uma descrição comportamental algorítmica, de uma estrutura no nível RT. Nesta estrutura constam unidades funcionais (como somadores), registradores e elementos de interconexão (barramentos ou multiplexadores). No entanto, estes elementos estruturais são "genéricos", isto é, não são instâncias de componentes concretos implementados em alguma tecnologia. A Figura 11 mostra a estrutura de controle genérica definida por esta metodologia.

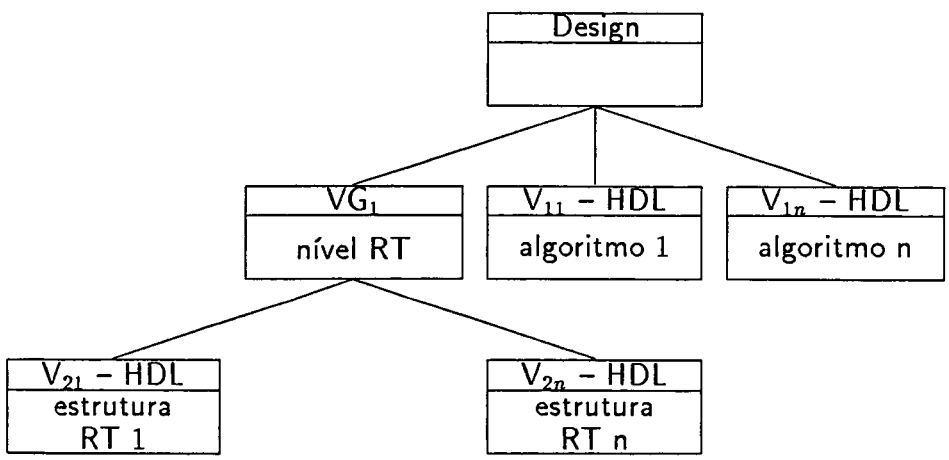


Figura 11: Estrutura de controle para o primeiro exemplo, metodologia inicial

Views HDL V_{1i} colocadas diretamente sob o Design contêm descrições algorítmicas alternativas. Um ViewGroup VG_1 colocado também sob o Design reune todas as Views que vierem a ser criadas contendo descrições estruturais no nível

RT. A decisão de se definir um ViewGroup em lugar de uma View foi tomada já em função de novas ferramentas que possam vir a ser integradas e que venham permitir uma extensão do projeto no nível RT, assim como para permitir um armazenamento mais eficiente dos Ports, como se verá mais adiante. Neste primeiro momento, estão definidas unicamente Views V_{2i} sob este ViewGroup, que contêm as descrições RT geradas pela síntese de alto nível. Estas Views são ainda do tipo HDL, pois a descrição RT não é composta por uma interconexão de instâncias de outros Designs. Views alternativas sob VG_1 podem conter por exemplo arquiteturas com barramentos e arquiteturas com multiplexadores.

Metodologia intermediária Sobre a metodologia inicial é definida uma nova metodologia, que acrescenta uma ferramenta que realiza o mapeamento entre a estrutura RT antes sintetizada e elementos concretos de uma dada tecnologia. Na Figura 12 é mostrada a estrutura de controle gerada pela hierarquia das duas metodologias iniciais.

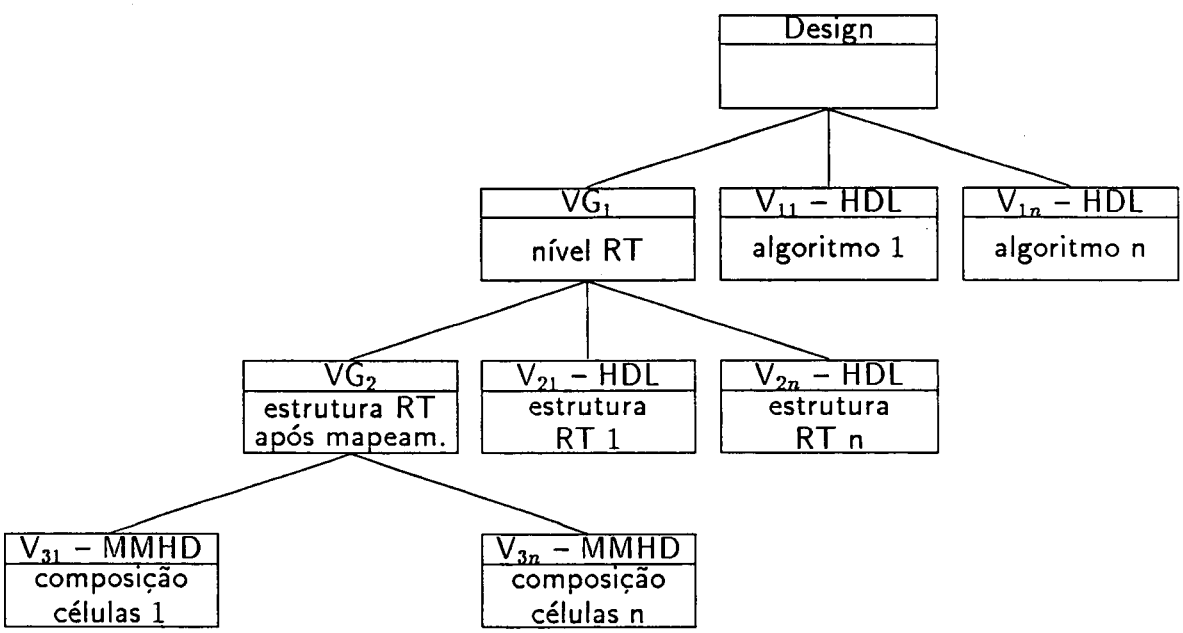


Figura 12: Estrutura de controle para o primeiro exemplo, metodologia intermediária

Um ViewGroup VG_2 foi acrescentado sob VG_1, tendo por objetivo reunir as descrições RT que já contiverem este mapeamento tecnológico. Sob este ViewGroup existem Views V_{3i}, do tipo MMHD, que contêm descrições RT com interconexões de instâncias de outros Designs já armazenados no Repositório. Diferentes Views sob VG_2 podem representar soluções com diferentes tecnologias (por exemplo CMOS dinâmico de 2 fases ou CMOS estático). O conjunto de Designs a serem instanciados em cada uma destas tecnologias (uma "biblioteca de células") poderia estar

armazenado numa Biblioteca específica para a tecnologia, à qual fosse associado um Processo. Uma restrição de integridade deveria verificar se todos os Designs instanciados num mesmo ViewState pertencem à mesma Biblioteca.

Metodologia final Uma metodologia final derivada das duas anteriores prevê a análise de testabilidade da estrutura RT projetada e, em função do resultado, uma ferramenta acrescenta determinadas estruturas de teste, de modo a tornar o circuito mais facilmente testável. A Figura 13 mostra a estrutura de controle resultante da hierarquia das três metodologias. Views V_{4i} de tipo MMHD foram acrescentadas sob VG_2, sendo nelas armazenadas as descrições RT já contendo as estruturas de teste. Cada V_{4i} pode representar uma diferente estratégia de teste (BILBO, Scan-Path, etc). As estruturas de teste são instâncias de Designs contidos em Bibliotecas específicas, às quais estão associados os mesmos Processos já mencionados anteriormente (CMOS estático, dinâmico 2 fases, etc).

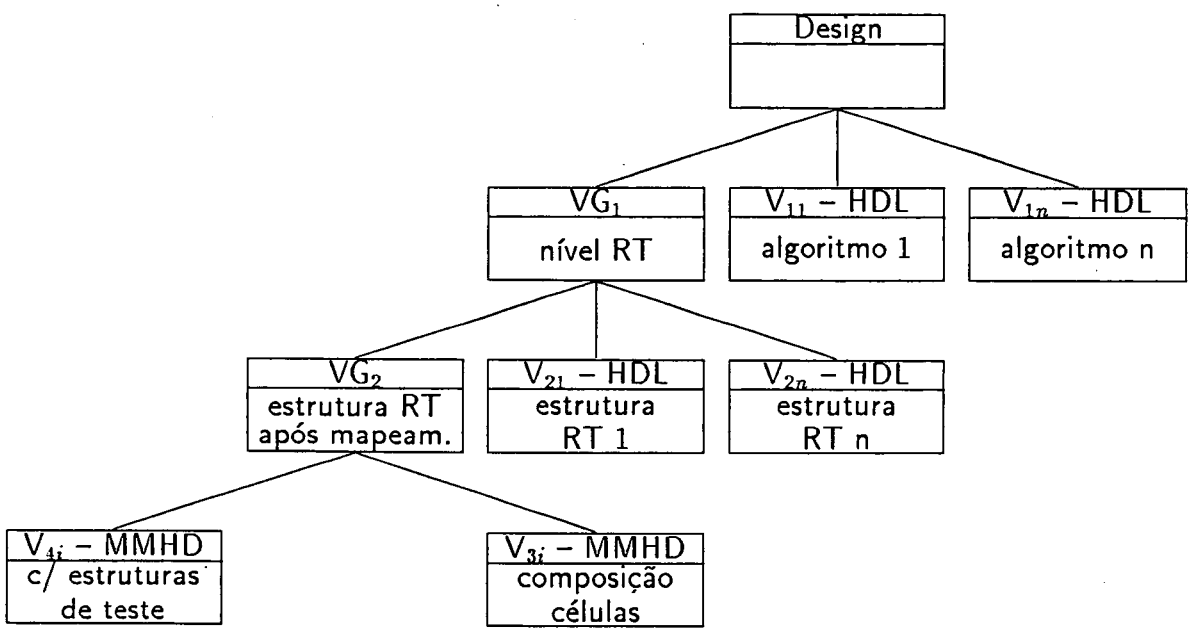


Figura 13: Estrutura de controle para o primeiro exemplo, metodologia final

Tratamento da interface Na metodologia básica, pode-se prever o armazenamento, no nodo raiz da estrutura de controle, dos Ports que serão certamente comuns a todos os ViewStates de um Design. Estes Ports, no início de um projeto *top-down*, são aqueles comuns aos diversos algoritmos a serem analisados. Cabe ao usuário determinar que Ports são estes para cada Design em particular. Ports eventualmente específicos para cada algoritmo são armazenados no nível das Views V_{1i} diretamente sob o Design.

De forma análoga, podem ser armazenados no nível das Views V_{2i} sob VG_1 os Ports que forem específicos da arquitetura RT correspondente a cada uma delas.

Já na metodologia intermediária, são armazenados no nível das Views V_{3i} sob VG_2 os Ports específicos da estrutura RT correspondente a cada uma delas. Aqui estão tipicamente os sinais de relógio necessários aos diferentes tipos de registradores (dinâmicos de 2 ou 4 fases ou estáticos).

Também a metodologia final acrescenta Ports que são específicos de cada estratégia de teste, correspondendo a sinais que permitem controlar e/ou observar sinais internos do objeto. Estes Ports são armazenados no nível das Views V_{4i} sob VG_2, que contêm as descrições RT já com as estruturas de teste.

Em [3] é proposta uma estrutura de controle genérica (mostrada na Figura 3) na qual ViewGroups reunem Views geradas para uma dada decisão de projeto, e não Views para um determinado nível de projeto, tal como neste exemplo. Aquela estrutura de controle favorece o armazenamento de Ports no nível dos ViewGroups, ao invés do armazenamento no nível das Views.

Correlações Na metodologia inicial, uma correlação deve representar, para cada View V_{2i}, a relação de equivalência desta com uma View V_{1i}. Note-se que, por outro lado, não é obrigatório que cada View V_{1i} esteja relacionada desta forma a uma View V_{2i}.

Correlações semelhantes devem ser associadas na metodologia intermediária a Views V_{3i}, relacionando-as a Views V_{2i}, e na metodologia final a Views V_{4i}, relacionando-as a Views V_{3i}.

Atributos Na metodologia final, uma ferramenta de análise obtém medidas de testabilidade das descrições RT estruturais armazenadas sob a View V_3. Cada medida contém valores de observabilidade e controlabilidade para sinais no interior de um objeto. A metodologia final deve então prever o acréscimo, em cada V_{3i}, de um atributo que armazene estas medidas. Note-se que V_3 havia sido definida pela metodologia intermediária, onde no entanto não havia sido previsto nenhum atributo relativo à testabilidade dos objetos.

Configurações Na metodologia final, uma restrição de integridade deve assegurar que as estruturas de teste (que são instâncias de determinados Designs) numa View V_{4i} sejam da mesma tecnologia que os Designs instanciados na View V_{3i} que deu origem a V_{4i}.

Objetos auxiliares Integrando-se à metodologia final ferramentas para a geração de vetores de teste, poderiam ser criados dois tipos de objetos auxiliares:

- *Casos de teste* seriam objetos gerados a partir das descrições algorítmicas iniciais (V_{1i}), contendo seqüências simbólicas de teste; correlações deveriam vincular cada *caso de teste* a uma V_{1i}.

- *Vetores de teste* seriam seqüências binárias de teste, geradas a partir dos *casos de teste* e das estruturas RT que já consideram o mapeamento tecnológico (V_{3i}); uma correlação deveria vincular cada *vetor de teste* a um *caso de teste* e a uma V_{3i}.

5.2 Metodologia AMPLO

Este exemplo, além de mostrar a definição de uma hierarquia de metodologias de projeto, irá ilustrar com mais detalhes o processo de integração de ferramentas de projeto.

5.2.1 Definição da hierarquia de metodologias de projeto

Este exemplo mostra a especificação de uma metodologia de projeto que emula aquela suportada no ambiente AMPLO [13,14], em desenvolvimento na UFRGS.

Metodologia básica A Figura 14 mostra a estrutura de controle genérica secundária, especificada sobre a estrutura genérica primária do GARDEN, que mapeia os conceitos de representação e gerência de dados suportados em AMPLO.

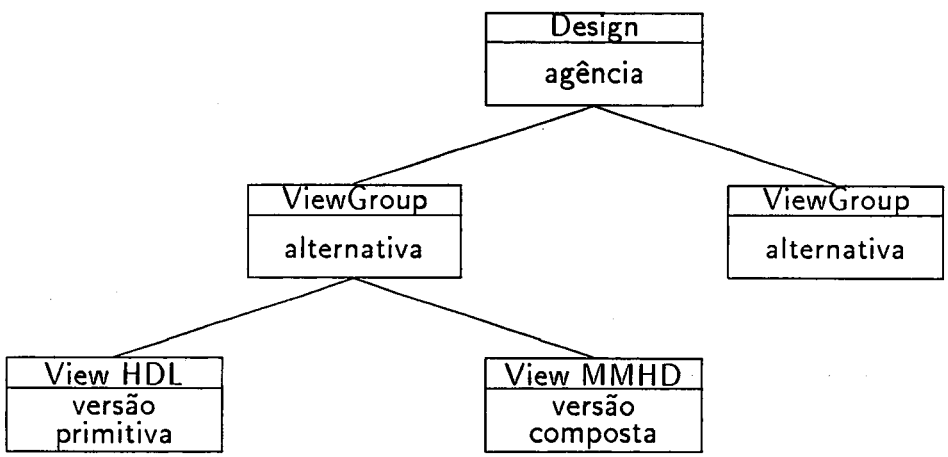


Figura 14: Estrutura de controle do sistema AMPLO mapeada para GARDEN

Designs correspondem a *agências.* Sob o Design há um número qualquer de ViewGroups que correspondem a *alternativas de projeto*, cada uma delas representando uma diferente definição para a interface do Design. Toda a descrição da interface é armazenada neste nível da estrutura de controle. Sob cada ViewGroup há um número qualquer de Views, correspondendo a *versões de projeto.* Há dois tipos de *versões.* As *versões primitivas* contêm descrições numa certa linguagem de descrição de hardware e não contêm instâncias com referências a outros Designs.

Elas são mapeadas para Views de tipo HDL. As *versões compostas* contêm interconexões de instâncias de outros Designs, sendo então mapeadas para Views de tipo MMHD. Modificações e Iterações não são utilizadas.

A interface de um objeto possui uma dada geometria para sua envoltória, definida através de dois atributos (dimensão horizontal e dimensão vertical) associados a cada ViewGroup. A interface é composta de um número qualquer de Ports. Cada Port possui quatro atributos: nome, tipo de dado, largura em bits e sentido.

Além da definição da topologia geral da estrutura de controle e dos atributos da interface, a metodologia de projeto especificada pela estrutura genérica secundária mostrada na Figura 14 deve restringir referências a objetos aos dois tipos de configurações existentes em *versões compostas* em AMPLO. Referências a ViewGroups (portanto a *alternativas de projeto*) criam *configurações dinâmicas*, enquanto referências a Views (portanto a *versões de projeto*) criam *configurações estáticas*. Referências a Designs não são permitidas em AMPLO. Referências a ViewStates não precisam ser consideradas já que a estrutura de controle não prevê ViewStates.

Metodologia intermediária Desta primeira estrutura de controle genérica secundária pode ser derivada uma outra estrutura secundária que acrescenta atributos que limitam o ambiente às três linguagens de descrição de hardware originalmente desenvolvidas no sistema AMPLO:

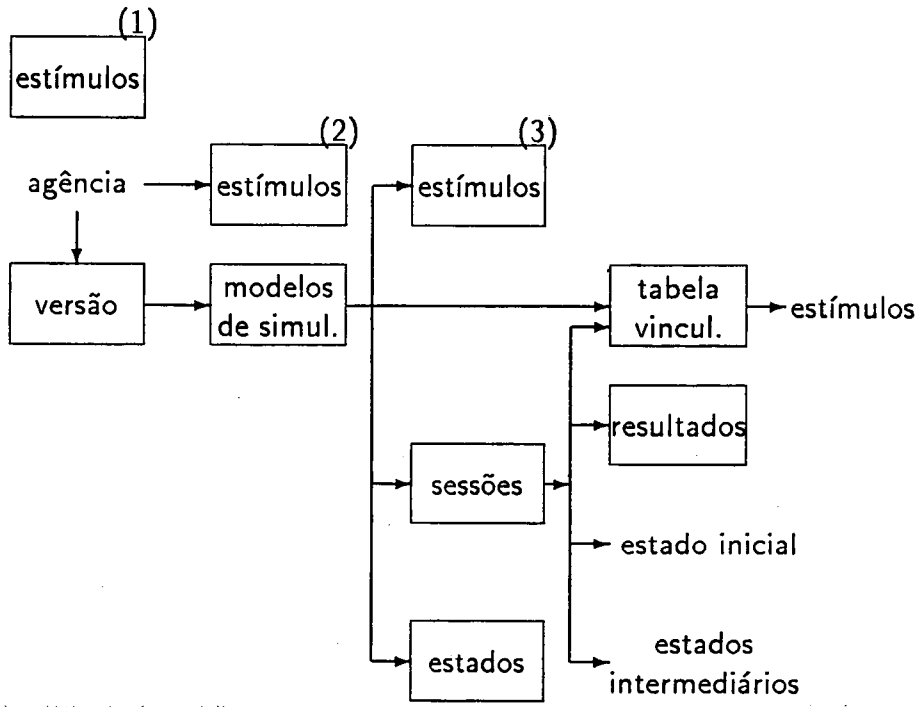
- Views HDL passam a ter um atributo ViewType cujo valor pode ser LAÇO, KAPA ou NILO (nomes das linguagens implementadas em AMPLO).
- Os tipos de dados associados aos Ports têm valores limitados a um conjunto pré-definido de tipos, dependente do valor do atributo ViewType. No caso do valor deste ser "KAPA", por exemplo, os tipos de dados possíveis para os Ports são "terminal", "bus" e "clock".

Nesta metodologia de projeto derivada, relações de equivalência podem ser estabelecidas entre Views de ViewTypes diferentes. A restrição de que os ViewTypes devem ser distintos nas Views correlacionadas deve poder ser estabelecida pela ferramenta de descrição da metodologia de projeto.

Metodologia final Em [15,16] é apresentado um ambiente específico de simulação implementado no sistema AMPLO. Este ambiente estende a estrutura de controle AMPLO com objetos auxiliares (*modelos de simulação, estímulos, tabelas de vinculação, sessões, estados* e *resultados*) e relações, mostrados na Figura 15, próprios para um suporte mais efetivo à gerência do processo de simulação.

Esta nova estrutura de controle genérica secundária é derivada da anterior pela inclusão de objetos auxiliares, correlações e restrições de integridade.

As correlações devem implementar todas as relações entre objetos auxiliares e destes com os objetos já existentes na estrutura de controle (ViewGroups e Views),



(1) Estímulos não associados com objetos
(2) Estímulos associados com agências
(3) Estímulos associados com modelos de simulação

Figura 15: Objetos do ambiente de simulação AMPLO

já que não há relações implícitas entre os objetos auxiliares, tal como as existentes entre Design, ViewGroups e Views.

Da mesma forma, há restrições de integridade intrínsecas na estrutura de controle primária que são automaticamente verificadas pelo sistema GARDEN, tais como restrições de referência ("um Componente não pode fazer referência a um objeto que não existe") e de composição ("um ViewGroup sempre deve estar associado a um Design"). No entanto, restrições relativas aos objetos auxiliares não serão verificadas automaticamente, e portanto deverão ser especificadas na metodologia de projeto. Exemplos de restrições são:

- "Um *estado intermediário* gerado numa *sessão* associada ao *modelo de simulação* M_i só pode ser usado como *estado inicial* de outra *sessão* para o mesmo M_i".
- "Uma *tabela de vinculação* só pode vincular a um *modelo de simulação* M_i *estímulos* associados a M_i ou à *agência* a partir da qual M_i foi construído, ou não associados a nenhum objeto".

5.2.2 Integração de ferramentas

O processo de integração de ferramentas será ilustrado através de três exemplos, demonstrando a utilização dos três métodos de integração propostos (forte, fraca seqüencial e fraca concorrente). As ferramentas escolhidas apresentam graus de interação diversos com a base de dados, justificando-se assim a escolha de um método de integração distinto para cada uma delas.

É importante neste exemplo notar-se que o modelo de dados AMPLO apresenta larga granularidade. *Agências* são objetos genéricos que em princípio contêm agregados de elementos ou funções mais primitivos. A granularidade do modelo também influencia a escolha do método de integração, conforme já mencionado na subseção 4.4.

A integração das ferramentas escolhidas irá também ilustrar a relação entre o processo de integração e a hierarquia de estruturas de controle genéricas secundárias, já que estas ferramentas estão associadas a diferentes níveis desta hierarquia.

Construtor de modelos O construtor de modelos é uma ferramenta que, a partir das descrições de *agências* contidas na base de dados, cria *modelos de simulação*, que são redes de *versões primitivas* (o que é uma exigência dos simuladores). Para esta construção é necessária a seguinte seqüência de passos:

- seleção de uma *agência* A;
- seleção de uma *alternativa* A_i para A;
- seleção de uma *versão* $A_{i,j}$ para A_i;
- se $A_{i,j}$ for uma *versão composta*, então, para cada componente de $A_{i,j}$, se ele for uma ocorrência de uma *alternativa* B_k de outra agência B, selecionar uma *versão* para B_k, i.e. repetir estes dois ultimos passos para o componente (se o componente for uma ocorrência de uma *versão*, a seleção não é necessária, mas de qualquer forma o último passo deve ser repetido sobre cada componente se a *versão* for composta).

Paralelamente a este processo de *configuração*, o construtor de modelos vai montando a rede de *versões primitivas* necessária à simulação. No final do processo, é criado na base de dados um objeto *modelo de simulação*.

É impossivel prever-se, antes do início da execução do construtor de modelos, quais *alternativas* e *versões* serão selecionadas pelo usuário, com o que torna-se impossível uma integração fraca seqüencial.

Além disto, o grau de interação desta ferramenta com a base de dados é bastante grande (sempre ressalvando que não tão intenso como aconteceria numa ferramenta operando sobre um modelo de dados com granularidade fina). As operações de busca de *alternativas* e *versões* na base de dados representam uma parcela considerável do tempo dispendido pela ferramenta, já que além destas existem apenas operações de exibição e seleção de opções na tela e montagem, pela ferramenta, do *modelo*

de simulação. Em virtude destas considerações, o método de integração forte é recomendável em relação a uma integração fraca concorrente.

A integração forte exige neste caso a re-implementação do código do construtor de modelos, substituindo funções como *busca versão* por operações relativas à estrutura de controle genérica GARDEN (*busca View*, no caso).

Tendo em vista que o construtor de modelos cria um objeto auxiliar definido no terceiro nível da hierarquia de metodologias de projeto, é neste nivel que ele deve ser integrado.

Editor de versões compostas O editor gráfico REDES destina-se à descrição de *versões compostas* de *agências.* Basicamente, o usuário realiza as seguintes tarefas:

- busca da base de dados uma *alternativa* A_i (i.e. uma definição de interface) para uma *agência* A, ou cria uma nova *alternativa* A_i para A (neste caso cria Ports e seus atributos);
- busca da base de dados uma *versão composta* $A_{i,j}$ para A_i, a partir da qual será criada, por edição, uma nova *versão composta* $A_{i,k}$, através das seguintes ações (opcionalmente o usuário pode criar uma nova *versão* a partir do zero):
  - cria componentes que são ocorrências de outras *agências* já previamente definidas na base de dados (pode se optar entre ocorrências de *alternativas* ou de *versões* destas *agências*);
  - coloca estas ocorrências em determinadas posições dentro do espaço delimitado para $A_{i,k}$ (i.e. define os valores de atributos geométricos de posicionamento para cada componente);
  - conecta Ports das interfaces dos componentes com outros Ports, sejam da interface de outros componentes, sejam da interface de $A_{i,k}$ (i.e. cria objetos Conexões com seus atributos geométricos de traçado).

À medida que novas *versões compostas* (e eventualmente também novas *alternativas*) vão sendo criadas, ocorrências delas podem ser utilizadas na descrição de novas *versões compostas*, sem que o usuário precise encerrar a sessão com o editor. Além disto, o usuário pode criar uma nova *agência* B, ou uma nova *alternativa* B_x de outra *agência* B já existente, durante a edição da *versão* $A_{i,k}$ da *agência* A, com o propósito de associar uma ocorrência de B_x a um componente de $A_{i,k}$. A nova alternativa B_x pode então ser criada antes que a edição de $A_{i,k}$ tenha sido completada, e sem que o usuário precise abandonar esta edição, mesmo que apenas temporariamente. Esta característica do editor REDES torna impossível uma integração fraca seqüencial, já que não é possível esperar o final de uma sessão de edição para efetuar as operações de atualização sobre a base de dados.

O editor REDES apresenta um grau de interação com a base de dados que é certamente mais baixo do que aquele do construtor de modelos. Durante uma edição, são executadas operações como *busca alternativa*, *busca versão*, *cria alternativa* e

cria versão. Como o usuário deve executar muitas outras ações interativas para a edição da *versão* (apontamento de objetos na tela, seleção de itens em cardápios, traçado de linhas, etc), o tempo gasto com o acesso à base de dados não é extremamente significativo. Em função disto, uma integração fraca concorrente seria possível sem trazer grande ineficiência.

Na descrição de integração do editor, cada função de acesso à base de dados AMPLO, como *busca versão*, seria mapeada para funções de acesso à estrutura de controle genérica GARDEN (*busca View*, no caso).

Tendo em vista que o editor REDES efetivamente implementado no sistema AMPLO é específico para o conjunto de linguagens LAÇO, KAPA e NILO, os tipos de dados associados aos Ports podem ter apenas os valores definidos nestas linguagens. O editor deve então ser integrado no segundo nível da hierarquia de metodologias de projeto, já que estes tipos de dados foram definidos neste nível. O editor também poderia ser integrado no terceiro nível, obviamente.

Pode ser facilmente imaginado um editor aberto em relação às linguagens disponíveis para a descrição das *versões primitivas.* Este editor poderia ser integrado no primeiro nível da hierarquia de estruturas genéricas secundárias. Os tipos de dados associáveis aos Ports seriam definidos pelas próprias ferramentas de projeto e seus valores não poderiam ser verificados pelo sistema de banco de dados.

Editor de versões primitivas Para cada uma das linguagens disponíveis para a descrição de *versões primitivas* existe um editor gráfico especializado. O usuário cria uma *versão primitiva* através das seguintes ações:

- busca da base de dados uma *alternativa* A_i (i.e. uma definição de interface) para uma *agência* A, ou cria uma nova *alternativa* A_i para A (neste caso cria Ports e seus atributos);
- busca da base de dados uma *versão primitiva* $A_{i,j}$ para A_i a partir da qual será criada, por edição, uma nova *versão primitiva* $A_{i,k}$, através de uma seqüência de operações que manipulam elementos particulares da linguagem em questão, p.ex. portas lógicas e interconexões entre elas, no caso da linguagem NILO (opcionalmente o usuário pode criar uma nova *versão* a partir do zero).

Neste caso, a criação da estrutura de elementos primitivos da linguagem não exige acessos ao banco de dados. Apenas no final da sessão será feito um único acesso à base de dados, sendo então criada a nova *versão primitiva.* No início da sessão terão sido feitos também alguns acessos à base de dados, correspondendo à busca de uma *alternativa*, e eventualmente à criação de uma nova *alternativa* e à busca de uma *versão primitiva* para edição.

Estes poucos acessos já são, pela própria natureza da ferramenta, realizados antes ou depois desta ser executada. Com isto, a integração fraca seqüencial é a mais apropriada, bastando para isto definir, na descrição de integração da ferramenta, a relação das *alternativas* e *versões* com os objetos da estrutura de controle GARDEN.

Da mesma forma que o editor REDES, um editor de *versões primitivas* depende do conhecimento dos tipos de dados particulares da linguagem para a qual ele é dedicado, de modo que ele deve ser integrado no segundo nível da hierarquia de metodologias de projeto.

5.3 Metodologias orientadas a arquiteturas

O exemplo a seguir ilustra a definição hierárquica de metodologias de projeto especializadas para determinados tipos de arquiteturas. O exemplo é baseado em [3], onde são definidas estruturas de controle para módulos dos tipos "Bloco Operacional" e "Bloco de Controle" através de metodologias M_{bo} e M_{bc}, respectivamente. Uma metodologia inicial M_0 define uma estrutura de controle que contém propriedades comuns a M_{bo} e M_{bc}.

Metodologia inicial A metodologia inicial M_0 define estruturas de controle como as mostradas na Figura 16. Para todo objeto a ser projetado é feita uma descrição algorítmica inicial, armazenada numa View V_a de tipo HDL, situada diretamente sob o Design. São então sintetizadas duas alternativas de projeto, uma para arquiteturas com barramentos e outra para arquiteturas com multiplexadores. Descrições para estas alternativas são agrupadas sob ViewGroups VG_b e VG_m, respectivamente. Para cada uma destas alternativas são testadas duas outras alternativas, uma delas para registradores dinâmicos de 2 fases e outra para registradores dinâmicos de 4 fases. Sob VG_b, por exemplo, descrições para estas novas alternativas são agrupadas sob ViewGroups VG_{b2} e VG_{b4}, respectivamente.

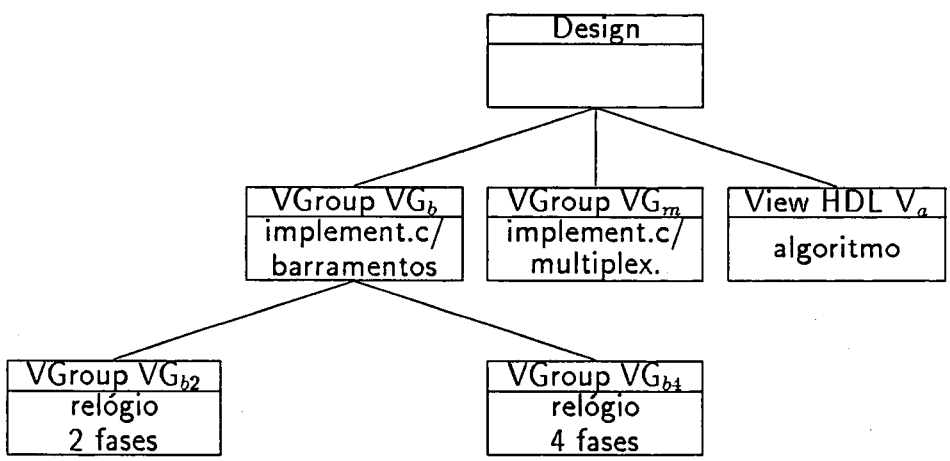


Figura 16: Estrutura de controle para M_0

Metodologia para blocos operacionais A Figura 17 mostra a estrutura genérica secundária definida pela metodologia M_{bo}, especializada para módulos de tipo "Bloco Operacional", a partir da metodologia M_0.

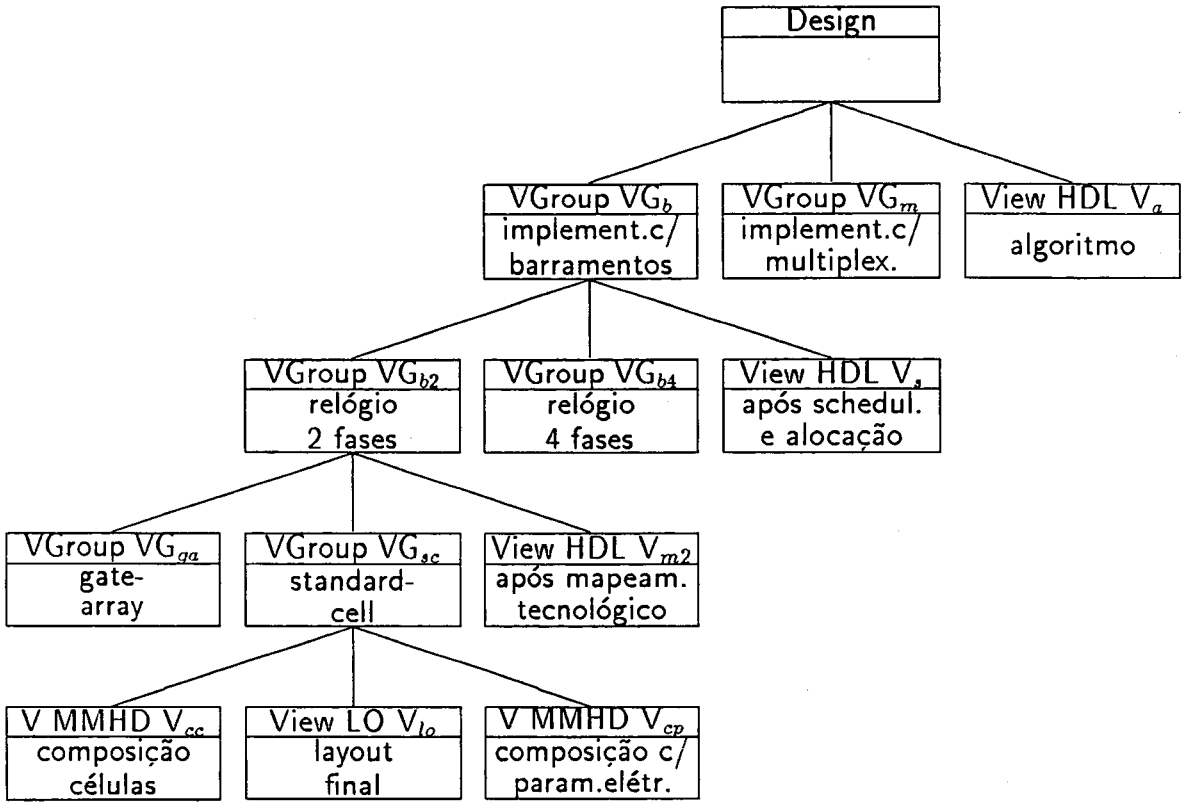


Figura 17: Estrutura de controle para M_{bo}

Esta metodologia M_{bo} acrescenta à estrutura gerada por M_0 os seguintes ViewGroups e Views:

- sob VG_b é incluída uma View HDL V_s que reune descrições RT sintetizadas de V_a por uma ferramenta que realiza o *scheduling* de operações e alocação de registradores, unidades funcionais e interconexões (uma inclusão análoga é feita sob VG_m);
- sob VG_{b2} são incluídos
  - uma View HDL V_{m2} que reune descrições RT sintetizadas de V_s por uma ferramenta que realiza o mapeamento tecnológico para registradores dinâmicos de 2 fases (uma inclusão análoga é feita sob VG_{b4});
  - um ViewGroup VG_{sc} que reune descrições sintetizadas a partir de V_{m2} para a estratégia *standard-cell*;
  - um ViewGroup VG_{ga} que reune descrições sintetizadas a partir de V_{m2} para a estratégia *gate-array*;
- sob VG_{sc} são incluídas

- uma View MMHD V_{cc} que reune descrições de composições de células de uma biblioteca, sintetizadas a partir de V_{m2} por uma ferramenta orientada para a estratégia *standard-cell*;
- uma View Layout (LO) V_{lo} que reune descrições de *layout*, sintetizadas a partir de V_{cc} por uma ferramenta de particionamento, posicionamento e roteamento para a estratégia *standard-cell*;
- uma View MMHD V_{cp} que reune descrições de composições de células de uma biblioteca, obtidas pela inclusão em V_{cc} de parâmetros elétricos obtidos a partir de V_{lo} por uma ferramenta de extração.

Sobre esta metodologia de projeto são integradas então as seguintes ferramentas:

- *scheduling* de operações e alocação de registradores, unidades funcionais e interconexões a partir de descrições algorítmicas de blocos operacionais;
- mapeamento tecnológico para registradores dinâmicos de 2 ou de 4 fases;
- síntese de composição de células para estratégias *standard-cell* e *gate-array*;
- particionamento, posicionamento e roteamento para as estratégias *standard-cell* e *gate-array*;
- extração de parâmetros elétricos.

Metodologia para blocos de controle A Figura 18 mostra a estrutura genérica secundária definida pela metodologia M_{bc}, especializada para módulos de tipo "Bloco de Controle", a partir da metodologia M_0.

Esta metodologia M_{bc} acrescenta à estrutura gerada por M_0 os seguintes ViewGroups e Views:

- sob VG_{b2} são incluídos
  - uma View HDL V_{maq} que reune descrições RT sintetizadas de V_a por uma ferramenta que extrai a máquina de estados do bloco de controle (uma inclusão análoga é feita sob VG_{b4});
  - um ViewGroup VG_{pla} que reune descrições sintetizadas a partir de V_{maq} para a estratégia PLA;
  - um ViewGroup VG_{la} que reune descrições sintetizadas a partir de V_{maq} para a estratégia lógica aleatória;
- sob VG_{la} são incluídas
  - uma View HDL V_{min} que reune descrições em termos de equações booleanas, obtidas por atribuição de estados e minimização lógica a partir de V_{maq}, voltadas para a estratégia lógica aleatória (analogamente é incluída uma View sob VG_{pla} para a estratégia PLA);

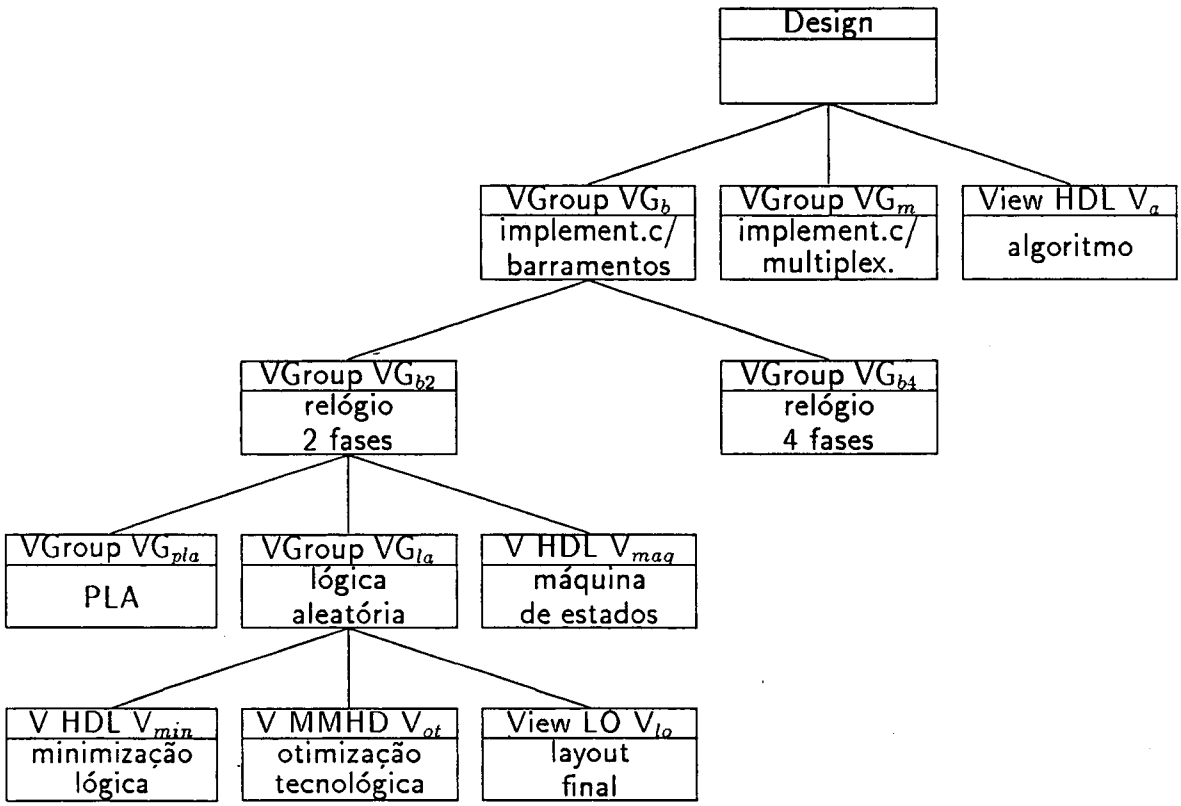


Figura 18: Estrutura de controle para M_{bc}

- uma View MMHD V_{ot} que reune descrições de composições de portas lógicas, obtidas pelo mapeamento e otimização tecnológica a partir de V_{min} (tal View não existe para a estratégia PLA, pois para ela o *layout* é gerado diretamente a partir das equações booleanas);
- uma View Layout (LO) V_{lo} que reune descrições de *layout*, sintetizadas a partir de V_{ot} por uma ferramenta de geração de *layout* para lógica aleatória (em estratégia *standard-cell* ou *gate-array*).

Sobre esta metodologia de projeto são integradas então as seguintes ferramentas:

- geração de máquina de estados a partir de um algoritmo;
- atribuição de estados e minimização lógica para as estratégias PLA e lógica aleatória;
- geração de *layout* a partir de equações booleanas, para a estratégia PLA;
- mapeamento e otimização tecnológica a partir de equações booleanas, para a estratégia lógica aleatória;
- geração de *layout* para lógica aleatória (em estratégia *standard-cell* ou *gate-array*).

5.4 Conclusões

Os exemplos detalhados nesta seção demonstram a aplicação do conceito de hierarquias de metodologias de projeto com três objetivos diversos:

- no segundo exemplo (metodologia AMPLO), as metodologias derivadas especializam o ambiente para determinadas ferramentas de projeto (linguagens, simuladores), sem que a estrutura de controle seja alterada;
- no primeiro exemplo (síntese de alto nível), as metodologias derivadas acrescentam níveis de ViewGroups e Views à estrutura de controle, especializando-a para ferramentas dedicadas a novos níveis de projeto;
- no terceiro exemplo (síntese de blocos operacionais e de controle), são geradas duas metodologias derivadas alternativas, dedicadas à síntese de diferentes módulos de um sistema digital; à semelhança do primeiro exemplo, estas metodologias acrescentam novos níveis à estrutura de controle.

Os três exemplos mostram também a integração de ferramentas em diferentes metodologias de uma mesma hierarquia de metodologias de projeto.

6 Comparação com outros trabalhos

Nesta seção serão discutidos outros trabalhos relacionados à gerência de metodologias de projeto, comparando suas soluções com aquelas adotadas no GARDEN. Em outros trabalhos a gerência de metodologias de projeto tem sido realizada principalmente através do controle da seqüência de ativação de ferramentas, sendo dada particular importância ao processo de integração das ferramentas de projeto. Antes desta análise, é necessário no entanto que se compare o conceito de gerência de metodologias de projeto adotado neste trabalho com o adotado em outros trabalhos.

6.1 Escopo da gerência de metodologias de projeto

Segundo a CFI – CAD Framework Initiative, um consórcio de empresas cujo objetivo é padronizar os principais aspectos de ambientes de projeto – uma metodologia de projeto [17] é "um conjunto seqüenciado de operações empregado na execução de uma determinada função", enquanto a gerência de metodologias de projeto "está relacionada à execução e controle de metodologias utilizadas no processo de projeto". Segundo a CFI, esta gerência pode ser decomposta em três funções principais:

- integração de ferramentas – como descrever ferramentas atômicas (requisitos de dados, definições de argumentos, caracterização do conjunto de comandos para o usuário, requisitos de recursos como CPU e memória);
- fluxo de tarefas – como descrever seqüências de tarefas (dependência entre tarefas, tarefas hierárquicas, controle de fluxo através de desvios condicionais, seleção e iteração, execução concorrente de tarefas);
- ambiente de execução – como executar as seqüências de tarefas, relacionando-as à invocação de ferramentas atômicas (seleção da melhor ferramenta para executar uma dada tarefa, invocação automática de uma ferramenta, armazenamento do e consulta ao histórico de execução de ferramentas, *backtracking* e recuperação de erros).

No sistema Ulysses [18], uma metodologia de projeto "determina como devem ser executadas diversas ferramentas de projeto, utilizando certos dados de projeto, de modo a cumprir com sucesso tarefas de projeto úteis". A metodologia de projeto especifica um plano de atividades que determina a seqüência de ferramentas a serem executadas e organiza o fluxo de dados entre as ferramentas, em termos de relações de entrada/saída entre ferramentas e dados.

Como se vê, estes conceitos de gerência de metodologias de projeto estão fortemente centrados no controle da seqüência de ativação de ferramentas, ao contrário do conceito adotado neste trabalho, que relaciona esta gerência principalmente à definição da estrutura de controle genérica suportada pelo ambiente.

Chiueh e Katz [19] definem como "gerência do processo de projeto" o "suporte à criação de evoluções de projeto alternativas, enquanto se mantém múltiplos con-

textos de projeto simultâneos, cada um com seus objetos de projeto e história de seqüências de operações". Por relacionar a gerência de metodologias de projeto à gerência de dados, que é justamente a função principal implementada pelas estruturas de controle do GARDEN, este conceito já está mais próximo daquele empregado neste trabalho.

No ambiente FACE [20], as funções da gerência de metodologias de projeto são "controlar todas as *views* de objetos criadas pelo ambiente, propagar alterações aos dados de modo a reconhecer quando os dados estão atualizados, e planejar a seqüência de execuções de ferramentas necessárias para a construção de uma certa *view*". Uma metodologia de projeto "define as regras para a execução de ferramentas e a construção de *views*", de modo que a "evolução de um objeto é restringida pela metodologia, que serve como um *template*" para a construção do objeto. Este conceito é extremamente similar àquele adotado neste trabalho, enfatizando o papel da metodologia de projeto como responsável pela definição da estrutura de controle associada a um objeto.

6.2 Controle da seqüência de ativação de ferramentas

O modelo de gerência de metodologias de projeto proposto neste relatório não aborda explicitamente a questão do controle da seqüência de ativação das ferramentas de projeto. No entanto, uma certa seqüência está obviamente implícita na geração de uma determinada estrutura de controle. Tome-se a Figura 18, por exemplo. Um ViewState sob a View MMHD V_{ot} deve ser gerado a partir de um ViewState sob a View HDL V_{min}. Isto significa que não pode haver uma View V_{ot} sob o ViewGroup VG_{la} sem que haja antes uma View V_{min} sob VG_{la}. Isto cria uma relação de precedência: a ferramenta de atribuição de estados e minimização lógica, que gera V_{min}, deve ser sempre executada antes da ferramenta de mapeamento e otimização tecnológica, que gera V_{ot}. Relações de equivalência estabelecidas entre os ViewStates destas Views, impostas através de restrições de integridade, podem forçar a seqüência: um ViewState sob V_{ot} só pode ser criado se, simultaneamente, for criada uma correlação com um ViewState sob V_{min} que estabeleça a relação de equivalência.

Neste relatório, ficou subentendida a sugestão de que a ferramenta de gerência de metodologias de projeto do GARDEN deveria especificar explicitamente a topologia da estrutura de controle, o que poderia ser feito através de uma ferramenta gráfico-interativa, por exemplo. Certas seqüências de ativação de ferramentas estariam sendo então implicitamente definidas através desta estrutura de controle. Certamente pode-se pensar numa solução alternativa: a seqüência de ativação das ferramentas é definida explicitamente. Em conjunto com a descrição de integração de cada ferramenta, que relaciona os objetos por ela manipulados com os objetos da base de dados, esta seqüência define implicitamente a estrutura de controle.

Pode-se também combinar estas duas técnicas num mesmo modelo de gerência

de metodologias, já que a definição explícita da estrutura de controle não impõe determinadas seqüências de ferramentas, mas apenas impede que outras seqüências sejam adotadas. Voltando ao exemplo mencionado anteriormente, não se consegue, unicamente definindo a topologia da estrutura de controle da Figura 18, exigir que seja *sempre* executada a ferramenta de mapeamento e otimização tecnológica quando a ferramenta de atribuição de estados e minimização lógica o é, já que nada obriga à existência de um novo ViewState sob V_{ot} para cada novo ViewState sob V_{min}.

Vários outros ambientes seguem pelo caminho da definição explícita das seqüências de ativação de ferramentas [18,19,20,21,22,23], como será brevemente descrito adiante.

Uma questão importante associada ao controle da seqüência de ativação de ferramentas em alguns ambientes [19,23] é o retorno a um ponto prévio do processo de projeto, quando o usuário (ou o próprio ambiente) conclui que enveredou por um caminho não satisfatório ou simplesmente deseja testar outra alternativa de projeto. Obviamente isto pode ser feito sob controle do usuário, se ele utilizar corretamente os recursos de versões e configurações disponíveis no ambiente. Para que este retorno possa ser administrado automaticamente pelo ambiente, no entanto, é necessário que este restaure (pelo menos do ponto de vista do usuário) o estado do banco de dados naquele ponto passado. Esta questão não foi considerada na proposta deste relatório, mas pode ser implementada em GARDEN, que oferece o recurso de TimeStamps para armazenar a história da evolução dos diversos objetos da base de dados.

Em [17] pode ser encontrada uma relação bastante detalhada de outras funções que devem ser atendidas por uma ferramenta de controle de seqüências de ativação de ferramentas. Estas funções, que não serão detalhadas aqui, são implementadas total ou parcialmente pelos ambientes antes referenciados, através de diferentes técnicas.

6.3 Exemplos de modelos de gerência de metodologias de projeto

Nesta subseção são brevemente apresentados os modelos de gerência de metodologias de projeto mais significativos descritos na literatura.

Ulysses Ulysses [18], desenvolvido pela Universidade de Carnegie-Mellon, foi o primeiro ambiente a considerar a questão do controle de metodologias de projeto. O ambiente é baseado num modelo de *blackboard*, uma base de dados onde estão armazenados dados de projeto manipulados pelas ferramentas e parâmetros para o *scheduling* das mesmas.

Cada ferramenta é uma "fonte de conhecimento" (KS – *knowledge source*) para o *blackboard*, ativada assincronamente sempre que os dados armazenados no *black-*

board atendem a determinadas "regras", também armazenadas na mesma base de dados. O *scheduler* é uma KS especial, implementada na forma de um sistema especialista, que monitora o *blackboard* e escolhe a próxima ferramenta a ser ativada, utilizando "parâmetros de resolução de conflitos" quando há várias KSs potencialmente executáveis. O RPOL (*Rating Policy Module*) é outra KS especial que calcula *ratings* para "pontos de projeto" ao longo do processo de projeto sempre que há atualizações no *blackboard*. O RPOL compara estes *ratings* periodicamente, chaveando o processo de projeto para o ponto mais promissor.

Todas as KSs, inclusive o *scheduler* e o RPOL, assim como as "regras" e os parâmetros para o *scheduling*, são descritos através de uma linguagem de alto nível especializada, denominada *scripts*.

Cadweld O ambiente Cadweld [21] também foi desenvolvido em Carnegie-Mellon, como uma evolução de Ulysses, utilizando ainda o conceito de *blackboard*. Em Cadweld, no entanto, não há um *scheduler* centralizado. Para cada ferramenta é construído um CTKO (*CAD Tool Knowledge Object*), que contém informações sobre as funcionalidades e capacidades da ferramenta e conjuntos de padrões de ativação aos quais a ferramenta pode responder. Cada CTKO monitora constantemente o *blackboard*. Quando há o ajustamento de um padrão aos dados armazenados no *blackboard*, a ferramenta "se candidata" para execução. A escolha da ferramenta a ser executada pode ser feita pelo usuário ou por uma ferramenta de controle, baseada nas informações sobre as ferramentas candidatas fornecidas por seus respectivos CTKOs.

Chiueh e Katz Em [19] é apresentado um "modelo de história de projeto" que estende trabalhos anteriores desenvolvidos na Universidade de Berkeley, o gerenciador de dados Oct [5] e um servidor de versões [24]. Uma "atividade" é um agrupamento de "tarefas" (invocações de ferramentas), ordenado no tempo na forma de uma árvore (uma tarefa pode ter várias tarefas como suas sucessoras na atividade). Um nodo nesta árvore define um "ponto de projeto", que identifica de forma unívoca um estado do banco de dados. Um "cursor" aponta para o ponto de projeto corrente. A árvore vai sendo construída à medida que o usuário realiza tarefas. A qualquer momento, o usuário pode retornar o cursor a um ponto de projeto passado, criando assim um novo ramo na árvore a partir daquele nodo. Neste novo ponto de projeto corrente, o usuário tem acesso apenas aos dados da base de dados naquele estado.

Este sistema permite que o usuário crie, modifique e re-utilize metodologias de projeto de forma interativa ao longo de um projeto. O conceito de metodologia de projeto nele implementado está bastante relacionado à gerência de dados de projeto [1]: os "pontos de projeto" fornecem uma funcionalidade realizada em outros sistemas por "configurações" (em Oct [25]) ou "TimeStamps" (em GARDEN [2]). No entanto, não é estabelecida uma relação com estruturas de controle associadas aos objetos (em outras palavras, a gerência de dados é realizada sem a utilização

explícita de estruturas de controle).

FACE No sistema FACE [20], desenvolvido pela GE, uma metodologia de projeto é representada por um grafo acíclico dirigido, onde os nodos são invocações de ferramentas e *views* dos objetos de projeto. O grafo mostra quais *views* são necessárias para a execução de uma ferramenta e quais são criadas pela ferramenta. Associados ao nodo que corresponde à invocação de uma ferramenta podem ser armazenados parâmetros necessários àquela invocação particular. O usuário pode definir metodologias de maneira gráfico-interativa. Conforme já mencionado anteriormente, este conceito de metodologia de projeto está fortemente vinculado à gerência de dados de projeto alcançada via estruturas de controle associadas aos objetos de projeto, tal como no método proposto neste relatório.

ADAM DPE (*Design Planning Engine*) [23] é um gerenciador de metodologias de projeto desenvolvido para o ambiente ADAM [26] da Universidade de Southern California. DPE é um sistema especialista que constrói, avalia e executa dinamicamente "planos de projeto". A partir de um estado inicial, e usando conhecimento sobre as ferramentas que podem ser aplicadas a este estado, DPE constrói um plano possível. A partir de estimativas para cada caminho alternativo no plano, obtidas rapidamente por ferramentas de avaliação, DPE escolhe um caminho a ser executado. Estimativas feitas a posteriori por ferramentas de avaliação mais precisas confirmam a escolha inicial ou levam DPE a retornar a um ponto anterior e escolher um novo caminho. A partir do novo estado obtido, DPE repete o processo de construção do plano.

À semelhança de Ulysses, e ao contrário dos demais ambientes, é o próprio DPE que, a partir de informações por ele obtidas, seleciona a próxima ferramenta para execução. Enquanto em Ulysses as informações utilizadas para a decisão são geradas por uma única ferramenta especializada que é executada periodicamente, o DPE ativa automaticamente ferramentas de avaliação estimativa específicas de cada estado do plano de projeto para obter as informações necessárias. À semelhança do trabalho de Berkeley, o DPE vai criando um plano sobre o qual se pode fazer *backtracking*. Naquele caso, no entanto, o retorno a um ponto anterior do processo é decidido pelo usuário, enquanto o DPE o faz automaticamente a partir de avaliações disparadas por ele mesmo.

7 Conclusões e trabalhos futuros

Este relatório propôs um modelo para a gerência de metodologias de projeto em ambientes de projeto de sistemas e circuitos eletrônicos. O modelo é específico para o ambiente GARDEN, aproveitando de maneira inovativa a propriedade original do modelo de dados suportado por este ambiente, pela qual um objeto pode ser representado através de uma hierarquia de ViewGroups (uma "estrutura de controle") de profundidade variável, podendo ser definido um critério qualquer para o agrupamento de Views sob os ViewGroups. Foi mostrado que a definição de uma metodologia de projeto no âmbito do GARDEN implica na definição de uma estrutura de controle genérica que serve de "molde" para todos os objetos que vierem a ser projetados.

O mecanismo de definição de metodologias de proposto utiliza esta propriedade de modo a obter uma dupla flexibilidade, não encontrada em outros ambientes:

- O modelo permite a definição hierárquica de estruturas de controle, de modo que uma nova estrutura de controle herde todas as propriedades da estrutura que é sua ascendente na hierarquia e acrescente novas extensões ou restrições a ela.
- Um projetista pode estar situado em qualquer ponto desta hierarquia de metodologias, tendo portanto maior ou menor flexibilidade para realizar suas tarefas.

O modelo proposto é apenas esboçado e justificado neste relatório. Ainda deve ser realizado um longo trabalho para o detalhamento dos mecanismos necessários à sua implementação e para a especificação da ferramenta que irá suportá-lo.

Vários trabalhos futuros podem ser realizados a partir das idéias aqui expostas. Em primeiro lugar, é necessária a especificação e construção de uma ferramenta que implemente o modelo de definição de metodologias de projeto proposto. A ferramenta deve oferecer ao construtor da aplicação e ao projetista (dois usuários que podem se confundir, conforme explicado na subseção 4.2) uma linguagem de alto nível para a especificação da metodologia e para a integração de ferramentas, evitando paradigmas pouco adequados a projetistas de sistemas eletrônicos, e deve suportar meios de referência aos objetos da estrutura de controle, tanto pelos projetistas como pelas ferramentas de projeto. Esta ferramenta deve implementar o conceito de herança entre metodologias, básico para o modelo.

Como sugestão inicial poderia ser implementada uma ferramenta interativa que permitisse a representação gráfica e a edição de estruturas de controle e a definição de suas propriedades. Na implementação de metodologias derivadas, esta ferramenta garantiria que o construtor da aplicação não alteraria propriedades da metodologia derivadora e que a extensão ou restrição à estrutura de controle genérica já existente seria feita de forma consistente.

Tendo em vista a complexidade desta ferramenta de gerência de metodologias de projeto, uma alternativa visando uma primeira implementação mais rápida seria

o suporte a um único nível de estrutura de controle genérica secundária. Teríamos então a abordagem clássica de um "modelo de dados semântico" (a estrutura de controle genérica primária do GARDEN) e um "esquema conceitual" (a estrutura secundária) para uma dada aplicação. Esta solução dispensa o mecanismo de herança de propriedades entre estruturas de controle hierarquizadas. O "esquema conceitual" poderia ser implementado através de seqüências de chamadas às funções que manipulam os objetos da estrutura de controle genérica primária.

Em qualquer uma destas duas soluções é ainda necessária a implementação de um mecanismo para a definição da integração das ferramentas de projeto. Conforme já sugerido na subseção 4.4, poderia se pensar numa linguagem única para a definição das metodologias de projeto e do processo de integração de ferramentas, pois na integração também é necessária a referência aos objetos da estrutura de controle. A linguagem única seria conveniente pois o processo de integração precisa ser consistente com a metodologia de projeto na qual a ferramenta será utilizada.

Foram propostos neste trabalho três métodos de integração de ferramentas:

- uma integração fraca, para estruturas de controle com larga granularidade, implementada através de um "mapeamento seqüencial", no qual o mapeamento entre os dados manipulados pela ferramenta e os objetos da estrutura de controle se dá antes e depois da execução da ferramenta;
- uma integração fraca, para estruturas de controle com granularidade fina, implementada através de um "mapeamento concorrente", no qual o mapeamento se dá durante a execução da ferramenta; e
- uma integração forte, para estruturas de controle com qualquer granularidade, na qual o mapeamento está implícito no código da ferramenta.

O segundo método de integração fraca é obviamente mais complexo do que o primeiro, sendo no entanto desejável e mesmo necessário para a integração eficiente de ferramentas com alto grau de interação com a base de dados, tal como ilustrado na seção 5.2. A implementação da ferramenta de gerência de metodologias de projeto deve portanto prever, desde o início, ambos os métodos de integração fraca. O método de integração forte é sempre possível, sendo desejável no desenvolvimento de novas ferramentas de projeto específicas para o ambiente.

Uma terceira alternativa que poderia ainda ser estudada para o uso do GARDEN é a não implementação de nenhuma estrutura de controle genérica secundária, com o que não se definiria nenhuma metodologia de projeto e não seria necessário um processo explícito de integração de ferramentas. Esta alternativa corresponde na prática ao método de integração forte, onde as ferramentas seriam implementadas diretamente sobre as funções de manipulação dos objetos da estrutura de controle primária. Esta solução obviamente deixa para as ferramentas de projeto um grande grau de liberdade na construção das estruturas de controle exemplares, liberdade esta que poderia ou não ser repassada ao projetista (ver subseção 4.5). Nesta

solução, se poderia na prática definir uma metodologia de projeto "implicitamente" através da implementação das ferramentas de projeto.

Na seção 6 foi mostrado que a gerência de metodologias de projeto não deve ficar restrita à definição da estrutura de controle genérica e à integração de ferramentas. Entre as demais funções necessárias, destaca-se particularmente a imposição de seqüências de ativação de ferramentas em função do processo de projeto. Soluções bastante sofisticadas têm sido propostas para este controle, tal como visto na subseção 6.3. O projeto e a implementação de uma ferramenta de gerência de metodologias de projeto no escopo do ambiente GARDEN não podem prescindir desta funcionalidade.

Também vinculado ao controle das seqüências de ativação de ferramentas está o recurso de retorno a um ponto anterior do projeto, com uma automática restauração do estado do banco de dados naquele ponto por parte do ambiente. Esta funcionalidade pode ser implementada no escopo da gerência de metodologias de projeto em GARDEN, valendo-se do recurso dos TimeStamps, que permitem o armazenamento do histórico da evolução dos objetos de projeto na base de dados.

O modelo de dados do GARDEN não prevê objetos **descrição de configuração** [1], tal como implementados na linguagem VHDL [7], que permitem armazenar as referências completas a outros objetos (no caso do GARDEN, até o nível de ViewState) selecionadas para os componentes de um objeto quando da utilização deste objeto por uma dada ferramenta. O armazenamento de descrições de configurações é necessário para a interpretação de resultados obtidos por ferramentas de análise ou de síntese e é muito útil quando associado ao retorno a um estado passado da base de dados (um conceito levemente distinto de configuração é utilizado no gerenciador Oct [25] com este objetivo). Sugere-se que, quando do projeto de uma ferramenta de gerência de metodologias de projeto para GARDEN, seja considerado este recurso.

Agradecimentos

Arnaldo H. Viegas de Lima contribuiu, através de inúmeras e frutíferas discussões, para o desenvolvimento do modelo de gerência de metodologias de projeto proposto neste trabalho.

Referências

[1] F.R. Wagner. *Modelos de Representação e Gerência de Dados em Ambientes de Projeto de Sistemas Digitais.* Technical Report CCR-121, IBM Rio Scientific Center, Rio de Janeiro, 1991.

[2] E.B. de la Quintana, G.O. Annarumma, and P. Molinari Neto. *GARDEN - the Design Data Interface.* Technical Report CCR-107, IBM Rio Scientific Center, Rio de Janeiro, 1990.

[3] F.R. Wagner. *Mapeamento de um Processo de Projeto de Circuitos VLSI para o Modelo de Dados do Ambiente GARDEN.* Technical Report CCR-109, IBM Rio Scientific Center, Rio de Janeiro, 1990.

[4] F.R. Wagner and A.H. Viegas de Lima. Design version management in the GARDEN framework. In *28th Design Automation Conference*, ACM/IEEE, 1991.

[5] D.S. Harrison et al. Data management and graphics editing in the Berkeley Design Environment. In *International Conference on Computer Aided Design*, IEEE, 1986.

[6] J.A. Mulle, K.R. Dittrich, and A.M. Kotz. Design management support by advanced database facilities. In F.J. Rammig, editor, *IFIP Workshop on Tool Integration and Design Environments*, North-Holland, 1988.

[7] *IEEE Standard VHDL Language Reference Manual.* IEEE, New York, 1988.

[8] N. Kraft. Embedded tool encapsulations. In F.J. Rammig and R. Waxman, editors, *2nd IFIP International Workshop on Electronic Design Automation Frameworks*, North-Holland, 1991.

[9] K. Groening et al. From tool encapsulation to tool integration. In F.J. Rammig and R. Waxman, editors, *2nd IFIP International Workshop on Electronic Design Automation Frameworks*, North-Holland, 1991.

[10] D. Gajski and R. Kuhn. Guest's editors introduction. *IEEE Computer*, December 1983.

[11] M.J. Chung and S. Kim. An object-oriented VHDL design environment. In *27th Design Automation Conference*, ACM/IEEE, 1990.

[12] D. Gedye and R.H. Katz. Browsing the chip design database. In *25th Design Automation Conference*, ACM/IEEE, 1988.

[13] F.R. Wagner, C. Freitas, and L.G. Golendziner. The AMPLO system - an integrated environment for digital systems design. In F.J. Rammig, editor, *IFIP Workshop on Tool Integration and Design Environments*, North-Holland, 1988.

[14] L.G. Golendziner and F.R. Wagner. Modeling digital systems as complex objects. In *9th International Symposium on Computer Hardware Description Languages and their Applications*, IFIP, 1989.

[15] F.R. Wagner et al. Recursos de gerência de projeto no sistema AMPLO. In *XVII Seminário Integrado de Software e Hardware*, SBC, Vitória, 1990.

[16] F.R. Wagner et al. Data management facilities in the AMPLO design framework. In F.J. Rammig and R. Waxman, editors, *2nd IFIP Workshop on Electronic Design Frameworks*, North-Holland, 1991.

[17] K.W. Fiduk, S. Kleinfeldt, M. Kosarchyn, and E.B. Perez. Design methodology management – a CAD Framework Initiative perspective –. In *27th Design Automation Conference*, ACM/IEEE, 1990.

[18] M.L. Bushnell and S.W. Director. VLSI CAD tool integration using the Ulysses environment. In *23rd Design Automation Conference*, ACM/IEEE, 1986.

[19] T. Chiueh and R.H. Katz. Managing the VLSI design process based on a structured history framework. In F.J. Rammig and R. Waxman, editors, *2nd IFIP International Workshop on Electronic Design Automation Frameworks*, North-Holland, 1991.

[20] W.D. Smith et al. FACE Core Environment: the model and its application in CAE/CAD tool development. In *26th Design Automation Conference*, ACM/IEEE, 1989.

[21] J. Daniell and S.W. Director. An object-oriented approach to CAD tool control within a design framework. In *26th Design Automation Conference*, ACM/IEEE, 1989.

[22] A. Di Janni. A monitor for complex CAD systems. In *23rd Design Automation Conference*, ACM/IEEE, 1986.

[23] D.W. Knapp and A.C. Parker. A design utility manager: the ADAM planning engine. In *23rd Design Automation Conference*, ACM/IEEE, 1986.

[24] R.H. Katz et al. Design version management. *IEEE Design & Test*, February 1987.

[25] M. Silva, D. Gedye, R.H. Katz, and A.R. Newton. Protection and versioning for Oct. In *26th Design Automation Conference*, ACM/IEEE, 1989.

[26] J. Granacki, D. Knapp, and A. Parker. The ADAM Advanced Design Automation System: overview, planner, and natural language interface. In *22nd Design Automation Conference*, ACM/IEEE, 1985.