UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

Rafael Luiz Cancian

UM MODELO EVOLUCIONÁRIO DE OTIMIZAÇÃO MULTIOBJETIVO PARA EXPLORAÇÃO DO ESPAÇO DE PROJETO EM SISTEMAS EMBARCADOS

Florianópolis(SC)
2011

Rafael Luiz Cancian

UM MODELO EVOLUCIONÁRIO DE OTIMIZAÇÃO MULTIOBJETIVO PARA EXPLORAÇÃO DO ESPAÇO DE PROJETO EM SISTEMAS EMBARCADOS

Trabalho apresentado ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas do Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina como requisito parcial para obtenção do grau de Doutor em Engenharia de Automação e Sistemas.

Orientador: Marcelo Ricardo Stemmer, Dr. Eng.
Co-orientador: Antônio Augusto Medeiros Fröhlich, Dr. Sc.

Florianópolis(SC)
2011

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

C215m Cancian, Rafael Luiz
Um modelo evolucionário de otimização multiobjetivo para exploração do espaço de projeto em sistemas embarcados [tese] / Rafael Luiz Cancian ; orientador, Marcelo Ricardo Stemmer. - Florianópolis, SC, 2011.
1 v.: grafs., tabs.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de sistemas. 2. Sistemas embutidos de computador. 3. Projetos de engenharia - Métodos de simulação. 4. Métodos orientados a objetos (Computação). 5. Algoritmos de computador. I. Stemmer, Marcelo Ricardo. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

CDU 621.3-231.2(021)

Rafael Luiz Cancian

UM MODELO EVOLUCIONÁRIO DE OTIMIZAÇÃO MULTIOBJETIVO PARA EXPLORAÇÃO DO ESPAÇO DE PROJETO EM SISTEMAS EMBARCADOS

Esta Tese de Doutorado foi julgada adequada para obtenção do Título de "Doutor" em Engenharia de Automação e Sistemas, Área de Concentração em Sistemas de Computação, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina.

Florianópolis(SC), 06 de Julho de 2011

_____

Prof. José Eduardo Ribeiro Cury, Dr. Eng.
Coordenador do Curso

_____

Prof. Marcelo Ricardo Stemmer, Dr. Eng.
Orientador

_____

Prof. Antônio Augusto Medeiros Fröhlich, Dr. Sc.
Co-orientador

Banca Examinadora:

_____

Prof. Marcelo Ricardo Stemmer, Dr. Eng.
Presidente da banca

_____

Prof. Ricardo Pezzuol Jacobi, Dr. Eng.
Universidade de Brasília - UnB

_____

Prof. Cesar Albenes Zeferino, Dr. Sc.
Universidade do Vale do Itajaí - UNIVALI

Prof. Leandro Buss Becker, Dr. Eng.
Universidade Federal de Santa Catarina - UFSC

Prof. Eduardo Augusto Bezerra, Dr. Eng.
Universidade Federal de Santa Catarina - UFSC

Prof. José Leonar Todesco, Dr. Eng.
Universidade Federal de Santa Catarina - UFSC

Dedico esta tese de doutorado à minha família, as pessoas mais importantes do mundo para mim, e em especial ao meu pai, Sérgio Gabriel Cancian, à minha mãe, Kacilda Yara Minotto Cancian, ao meu irmão, Felipe Augusto Minotto Cancian, e à minha amada esposa, Maiara Heil Cancian.

Dedico também esta tese a todas as pessoas de bem, contemporâneas e antepassadas, que dedicaram suas vidas ao desenvolvimento e divulgação do pensamento científico, crítico e racional, essenciais ao desenvolvimento e progresso da humanidade.

AGRADECIMENTOS

Agradeço inicialmente à Universidade Federal de Santa Catarina e a seus professores e servidores, em especial aos do Programa de Pós-Graduação em Engenharia de Automação e Sistemas, pelo apoio Institutional à realização deste doutorado. Agradeço também:

Ao meu orientador, professor Dr. Marcelo Ricardo Stemmer, e ao meu coorientador, professor Dr. Antônio Augusto Medeiros Fröhlich, pelo apoio, confiança, auxílio e oportunidade oferecidos.

Aos integrantes do Laboratório de Integração de Software e Hardware, em especial a Arliones Hoeller Junior, Giovani Graciolli, Hugo Marcondes e Lucas Warner, que auxiliaram na solução de vários problemas técnicos.

À minha família, pela educação, apoio e amor incondicionais, que me tornaram um homem bom e honesto, e que conclui com esta tese de doutorado o maior grau de escolaridade do sistema educacional.

Poderá ter sido assim que surgiram as primeiras células vivas. Os replicadores começaram não só a existir, mas também a construir invólucros, veículos para a preservação da sua existência. Os replicadores que sobreviveram foram aqueles que construíram máquinas de sobrevivência dentro das quais pudessem viver. As primeiras máquinas de sobrevivência terão sido, provavelmente, constituídas apenas por um revestimento protetor. Mas sobreviver foi se tornando cada vez mais difícil, à medida que apareciam novos rivais, com máquinas de sobrevivência melhores e mais eficientes. As máquinas de sobrevivência tornaram-se maiores e mais elaboradas e o processo foi cumulativo e progressivo. Teria de haver algum fim para o aperfeiçoamento gradual das técnicas e dos artifícios usados pelos replicadores para assegurarem a sua própria continuidade no mundo? Haveria tempo de sobra para aperfeiçoamentos. Que máquinas bizarras de autopreservação trariam consigo os milênios seguintes? Qual seria o destino dos primeiros replicadores 4000 milhões de anos depois? Não se extinguiram, pois são mestres antigos das artes de sobrevivência. Mas não esteja à espera de os encontrar flutuando livremente no mar; há muito tempo que eles abandonaram essa sua liberdade de cavaleiros andantes. Agora agrupam-se em colônias imensas, seguros no interior de gigantescos robôs desajeitados, afastados do mundo exterior, comunicando-se com ele através de vias indiretas e tortuosas, manipulando-o por controle remoto. Estão dentro de si e de mim; criaram-nos de corpo e mente; e a sua preservação é a razão última da nossa existência. Percorreram um longo caminho, esses replicadores. Agora respondem pelo nome de genes e as suas máquinas de sobrevivência somos nós.

—RICHARD DAWKINS (The Selfish Gene, 1976)

RESUMO

O projeto de sistemas embarcados tem se tornado mais complexo à medida em que ocorrem avanços na tecnologia e nas aplicações, forçando novas abordagens e metodologias de projeto. Em praticamente todas as metodologias modernas a etapa de exploração do espaço de projeto tem merecido destaque, pois é a responsável por gerar e analisar diferentes possíveis soluções de projeto e selecionar a melhor. A exploração do espaço de projeto é, então, um problema de otimização multiobjetivo em que o conjunto de possíveis soluções costuma ser enorme, caso em que técnicas heurísticas como os algoritmos evolucionários têm recebido grande destaque. Nesta tese foram desenvolvidos metamodelos que representam o projeto de sistemas embarcados pela metodologia de sistemas dirigidos pela aplicação (ADESD) e seus componentes lógicos e físicos. Esses metamodelos foram mapeados a um novo modelo evolucionário com modificações bioinspiradas que é utilizado para otimização multiobjetivo e, assim, para a exploração do espaço de projeto em sistemas embarcados. A exploração hierárquica, a representação e evolução tanto dos suportes de hardware físico quanto sintetizável, as modificações bioinspiradas incluídas no modelo evolucionário e sua avaliação usando indicadores e conjuntos de teste consagrados correspondem às principais contribuições desta tese. Os resultados demonstram a viabilidade do modelo desenvolvido para exploração do espaço de projeto no contexto proposto e um aumento da qualidade das soluções encontradas em alguns problemas de teste, com consequente aumento do sobrecusto computacional.

Palavras-chave: Sistemas Embarcados. Exploração do Espaço de Projeto. Projeto de Sistemas Dirigidos pela Aplicação. Otimização Multiobjetivo. Algoritmos Evolucionários.

ABSTRACT

The design of embedded systems has become more complex as technology and applications evolve, forcing new design approaches and methodologies. In almost all modern methodologies, the design space exploration has been highlighted because it is responsible for generating and analyzing different possible design solutions and to select the best one. The design space exploration is a multi-objective optimization problem on which the set of possible solutions is usually huge. Therefore, heuristic techniques, such as evolutionary algorithms has received great attention in this area. This thesis describes meta-models that represent the design of Application Driven Embedded Systems (ADESD) and their components. Meta-models are mapped to an evolutionary model with biological inspired modifications which is used for multi-objective optimization, and thus to explore the design space of embedded systems. The modifications included into the evolutionary model to support the ADESD design methodology compose part of the contributions of this thesis, as well the optimization performance evaluation of such modifications using the weighted hypervolume indicator and well established testing problems sets. Results show the model can be used for design space exploration in the proposed context and that it increases the quality of non-dominated solutions for some problem tests, with the drawback of increasing computational overhead.

Keywords: Embedded Systems. Design Space Exploration. Multi-objective Optimization. Evolutionary Algorithms.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

LISTA DE SÍMBOLOS

LISTA DE EQUAÇÕES

SUMÁRIO

PARTE I

Introdução

CAPÍTULO 1

INTRODUÇÃO

Numa primeira etapa a janela aberta pela ciência faz-nos tremer, ao nos retirar o calor que vem dos tradicionais mitos humanos, mas depois o ar fresco traz-nos vigor, e os grandes espaços têm um esplendor muito próprio.
—BERTRAND RUSSEL (What I Believe, 1925)

A primeira parte desta tese é composta por este capítulo introdutório que é dedicado à apresentação do trabalho desenvolvido, incluindo uma descrição geral, os problemas tratados, uma síntese dos resultados obtidos e a forma de organização do texto. A seção 1.1 contextualiza esta tese e os principais temas envolvidos, que compreendem os sistemas embarcados e a exploração de seu espaço de projeto, a otimização multiobjetivo associada a essa exploração e a técnica de otimização utilizada, que é a dos algoritmos evolucionários. A seção 1.2 resume os problemas tratados nesta tese e introduz a solução proposta. Na seção 1.3 os objetivos principais e específicos são explicitados, e na seção 1.4 é delimitado o escopo da tese e são apresentados resumidamente alguns resultados e limitações. Por fim, a seção 1.5 fornece a estrutura geral desta tese e descreve sucintamente o que será tratado em cada capítulo seguinte.

1.1 Contextualização

Há tempos os sistemas computacionais deixaram de existir apenas sob a forma de grandes computadores e *desktops* e passaram a existir em praticamente qualquer equipamento de uso cotidiano sob a forma de sistemas computacionais embarcados, ou simplesmente sistemas embarcados. Os sistemas embarcados, inicialmente bastante simples e rudimentares, foram impulsionados por avanços na tecnologia de circuitos integrados, que proveram maior capacidade de processamento, de armazenamento e outras possibilidades advindas da reconfiguração de hardware. Com isso, as aplicações tornaram-se sucessivamente mais complexas, e tal complexidade exigiu uso de metodologias e técnicas de projeto, e seu aperfeiçoamento, que prossegue continuamente. Uma das técnicas mais consagradas para tratar a complexidade dos sistemas embarcados é a do desenvolvimento baseado em componentes, em que os sistemas são vistos como agregados

de componentes de software e hardware que são reutilizáveis em diferentes aplicações.

No decorrer das últimas décadas, muitas metodologias para o projeto e desenvolvimento de sistemas embarcados foram propostas, destacando-se as que utilizam o desenvolvimento baseado em componente e que consideram o projeto conjunto de (componentes de) hardware e software. Uma característica comum nessas metodologias é que a complexidade das tarefas estipuladas por elas exige o suporte de ferramentas computacionais, o que também reduz o tempo de projeto, automatiza a execução dessas tarefas e diminui a probabilidade de erros humanos. Uma das metodologias baseadas em componentes mais difundidas é a PbD (PbD - *Platform Based Design*). Essa metodologia de projeto guia o desenvolvimento de plataformas de hardware reusáveis a várias aplicações, abstraindo as diferenças entre várias dessas plataformas e criando uma única interface que pode ser vista pelas aplicações embarcadas, denominada *plataforma de sistema* [Keutzer et al. 2000].

Outra metodologia para o projeto e desenvolvimento de sistemas embarcados baseados em componentes é a ADESD (ADESD - *Application Driven Embedded System Design*), proposta por Fröhlich [Fröhlich 2001]. A ADESD ainda é pouco difundida, mas tem produzido resultados promissores que a tornam um bom alvo para pesquisas. Essa metodologia permite a criação de sistemas dirigidos pela aplicação, que são projetados de forma a satisfazerem exatamente os requisitos de aplicações-alvo, com o mínimo de sobrecusto [Fröhlich 2001]. Ela abstrai componentes de hardware e também de software através de interfaces bem definidas, e utiliza modernas técnicas de engenharia de software e de implementação para adaptar e configurar esses componentes a um cenário específico de execução. Esta tese visa contribuir com as ferramentas de suporte dessa metodologia, automatizando uma de suas tarefas: a exploração do espaço de projeto.

Diferentes metodologias definem diferentes tarefas, técnicas e fluxos para o projeto de um sistema embarcado. Porém, uma das etapas comuns a praticamente todas as metodologias modernas é a exploração do espaço de projeto (DSE - *Design Space Exploration*), dada sua grande importância na determinação de uma boa solução para o sistema final. A exploração do espaço de projeto é responsável por testar diferentes soluções alternativas e escolher a mais adequada a um projeto específico, de acordo com os requisitos da aplicação e do projetista. Para melhor compreensão do que deve ser realizado nessa etapa, considere o seguinte exemplo de projeto de um sistema embarcado para um tocador portátil de músicas

(estilo iPod).

O funcionamento básico da aplicação de um tocador portátil de músicas consiste em obter uma lista de músicas de um dispositivo de armazenamento não volátil (ex: memória Flash) e tocar cada uma dessas músicas sequencialmente. O usuário pode apenas executar os comandos básicos encontrados nesse tipo de equipamento: *play, pause, stop, forward, backward,* aumentar e diminuir o volume. Para esse projeto, os requisitos da aplicação podem consistir na existência de componentes que forneçam os serviços de busca e recuperação de arquivos armazenados, decodificação de músicas num formato específico (ex: MP3), reprodução sonora dessas músicas e também obtenção do estado de botões externos (dispositivos de entrada de comandos do usuário). Os requisitos de projeto podem incluir uma lista de objetivos, como minimizar o consumo de energia, o tamanho e o custo do equipamento. Os requisitos de projeto também podem incluir uma lista de restrições sobre esses objetivos e possivelmente outros, como consumo de energia menor que 1mA, tamanho menor que 20x35x15mm, custo de componentes menor que R$80,00 e frequência de operação menor que 300MHz, por exemplo.

Para esse projeto, soluções alternativas podem incluir diferentes: (1) Componentes de software que disponibilizem a funcionalidade solicitada pela aplicação, ou diferentes componentes de software dos quais eles recursivamente dependem; (2) Configurações (parâmetros) dos componentes de software escolhidos; (3) Componentes de hardware associados a alguns componentes de software (como os dispositivos de hardware associados aos seus *drivers*), ou diferentes componentes de hardware dos quais eles recursivamente dependem; (4) Configurações (parâmetros) dos componentes de hardware escolhidos (ex: frequência, largura dos barramentos); (5) Funcionalidades que podem ser executadas por componentes em software ou em hardware –particionamento hardware/software– (como a decodificação de áudio); (6) Elementos processadores nas quais executarão os componentes de software (processadores, microcontroladores, *softcores*, etc); e (7) Formas de interconexão entre os componentes de hardware, que define um suporte de hardware para o equipamento. Utilizando técnicas como co-simulação[1] ou outras análogas, essa etapa deve também estimar a eficiência de cada possível solução gerada em relação aos objetivos que foram definidos para o projeto, como minimizar o consumo de energia e as dimensões físicas. Por fim,

[1] a *co-simulação* corresponde à simulação integrada de software e hardware. Juntamente com outras técnicas de simulação analítica e modelagem matemática, são usadas para extrair informações de um sistema embarcado.

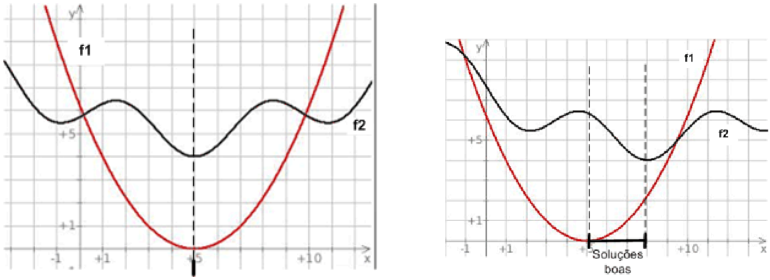
conhecendo a eficiência de diferentes soluções, deve guiar sua busca por uma solução ótima.

Num projeto qualquer, a exploração de soluções alternativas pode ocorrer em diferentes níveis de abstração, desde a escolha da quantidade de tarefas e sua alocação a processadores (nível de sistema), passando pela escolha de técnicas de compilação que podem beneficiar tempo de execução ou espaço de memória (nível de compilação), até o nível de projeto lógico, em que são escolhidas largura de barramentos e de dados e outras decisões que podem beneficiar área de silício, latência ou consumo de energia. Um projeto hipotético como o tocador portátil de músicas apresentado, com exploração unicamente no nível de componentes, que contenha apenas 8 componentes, em que cada componente tem apenas 2 parâmetros com apenas 4 valores possíveis para cada parâmetro, gera $8^{(2.4)} = 1,677.10^7$possíveis soluções diferentes. Pelo exemplo apresentado, percebe-se que a quantidade de possíveis soluções em sistemas reais pode ser enorme. Aliado ao fato da avaliação de cada solução pode consumir um tempo considerável, e que as funções matemáticas que descrevem as características de interesse do sistema (consumo de energia, tempo de execução, tamanho, etc) normalmente são desconhecidas, conclui-se que a exploração do espaço de projeto é uma tarefa complexa e que, em última análise, corresponde a um problema de otimização multiobjetivo.

Problemas de otimização multiobjetivo visam minimizar mais de uma função-objetivo simultaneamente. Embora também seja comum querer maximizar algumas funções, pode-se transformar o problema de maximização num problema de minimização[2], de forma que, sem perda de generalidade, pode-se afirmar que se trata de um problema de minimização [Benedetti, Farina e Gobbi 2006]. Contudo, é muito provável que não exista um único conjunto de variáveis que minimize todas as funções-objetivo simultaneamente, ou seja, não existe uma única solução ótima para todas as funções. A inexistência de uma única solução é reforçada na maioria dos problemas reais em que as funções-objetivo são conflitantes [Said, Bechikh e Ghédira 2010, Lara et al. 2010], como simultaneamente minimizar o consumo de energia e maximizar o desempenho, por exemplo. A figura 1.1 apresenta estas condições no caso de duas funções. Na figura 1.1(a), coincidentemente, existe uma única solução ótima (marcada pela linha tracejada) que minimiza simultaneamente as duas funções. Entretanto, o caso mais comum é o da figura 1.1(b), em que não há

[2] a maximização de uma função f_i é equivalente à minimização da função $-f_i$.

uma solução ótima, mas um conjunto de boas soluções. Por isso, os problemas de otimização multiobjetivo buscam um conjunto de soluções que representam a melhor relação (*trade-off*) entre as diferentes funções-objetivos [Coello 2005], conhecido como Conjunto de Pareto. As soluções nesse conjunto são ditas Pareto-dominantes, e correspondem às soluções em que não é possível melhorar o valor (diminuir, no caso de minimização) de qualquer função-objetivo sem piorar o valor de outra função-objetivo. Entretanto, encontrar soluções com dominância de Pareto é custoso ou mesmo impossível para muitas técnicas de otimização tradicionais [Coello 2006]. O problema também piora com o aumento da quantidade de funções-objetivo a serem minimizadas, e diversas dominâncias alternativas têm sido propostas. Assim, técnicas envolvendo heurísticas estocásticas têm merecido muito destaque nessa área e tem sido cada vez mais usadas. Entre essas técnicas estocásticas destaca-se a dos algoritmos evolucionários (EA - *Evolutionary Algorithms*).



(a) Existência de uma solução ótima (b) Inexistência de uma solução ótima

Figura 1.1: *Exemplo de funções para otimização multiobjetivo. Existência ou inexistência de uma solução ótima que minimiza simultaneamente ambas as funções-objetivo de um problema de otimização multiobjetivo. Em geral, as funções-objetivo são conflitantes, de modo que não existe uma solução ótima que as minimiza simultaneamente.*

Algoritmos evolucionários correspondem a uma classe de técnicas de otimização com inspiração biológica, como a evolução por seleção natural de Darwin e a genética de Mendel. Os algoritmos evolucionários têm sido usados há mais de uma década para a otimização multiobjetivo pois apresentam várias características interessantes, dentre as quais estão o fato de serem altamente paralelizáveis computacionalmente, de permitirem a exploração em dife-

rentes pontos do espaço de projeto numa única iteração e de serem aplicáveis a inúmeros problemas práticos de otimização, sem exigir conhecimentos específicos sobre características das funções sendo otimizadas [Xie e Ding 2009, Said, Bechikh e Ghédira 2010].

Certamente, a aplicação dos algoritmos evolucionários a problemas de otimização multiobjetivo exige sua adaptação a esse cenário, como o uso de operadores de seleção que considerem o conjunto de Pareto (ou alguma dominância alternativa) na escolha dos melhores indivíduos, visando obter soluções mais próximas desse conjunto e também melhor distribuídas [Beume et al. 2009]. Na literatura podem ser encontradas dezenas de variações nas heurísticas, incluindo novas estruturas e operadores genéticos que foram criados para a otimização multiobjetivo, bem como a inclusão de outros modelos de inspiração biológica, como nichos, sub-populações e migrações. Para avaliar o quão eficiente é cada uma dessas variações nas heurísticas, é necessário o uso de indicadores de qualidade adequados e de problemas de otimização conhecidos que possam ser usados como testes (*benchmarks*). Na literatura também é possível encontrar muitos indicadores e problemas de testes, mas a maioria deles ainda é voltada apenas a problemas com dois ou três objetivos, e muitos indicadores ainda são não-confiáveis, podendo fornecer resultados contraditórios em relação ao resultado do indicador e a proximidade das soluções ao Conjunto de Pareto. Recentemente, tem recebido ênfase o indicador de hipervolume, que agrega medidas de proximidade e distribuição das soluções num único valor [Bradstreet, While e Barone 2008, Zou et al. 2008, Beume et al. 2009, Auger et al. 2009]. Também tem recebido ênfase o uso de *frameworks* para geração de conjuntos de problemas de teste, que permitem gerar *benchmarks* escaláveis, com fronteira de Pareto conhecida e ainda "controlar" os obstáculos submetidos aos otimizadores sendo testados.

Vários elementos de inspiração biológica foram incluídos nos algoritmos evolucionários ao longo das décadas, mas parece que muitos referem-se ao processo macroscópico de evolução e de seleção, como aqueles citados no parágrafo anterior (nichos, sub-populações, migrações). Tem sido menos frequente a inclusão de elementos relacionados ao processo microscópico (genético e bioquímico) por trás da evolução. De forma geral, um indivíduo ainda é modelado de forma praticamente idêntica à que foi proposta há mais de 30 anos, ou seja, como um único cromossomo de tamanho fixo composto unicamente por genes também de tamanho fixo que sempre são decodificados para formar diretamente o fenótipo do indivíduo, que define totalmente sua aptidão à sobrevivência e à reprodução. A codificação das variáveis do problema nos genes ou é direta (nos algoritmos

genéticos) ou praticamente direta (na programação genética), de modo que, do genótipo, extrai-se diretamente o fenótipo. As taxas de mutação e de recombinação muito frequentemente ainda são fixadas empiricamente pelo projetista e são idênticas para todos os indivíduos e para todos os genes, que são sempre expressos para formar o fenótipo do indivíduo. A realidade biológica de onde esse modelo se inspira é muito diferente.

Como qualquer modelo, os algoritmos evolucionários abstraem parte da complexidade do sistema real e representam apenas os componentes e comportamentos mais importantes. Entretanto, dado que o processo bioquímico é a base sobre a qual a evolução atua[3], a abstração indevida de componentes importantes desse processo pode prejudicar a evolução. De modo análogo, a representação de componentes secundários ou irrelevantes pode aumentar sua complexidade e consumo de recursos computacionais sem trazer benefícios adicionais ao que o modelo se destina e que, no caso desta tese, é a otimização multiobjetivo. Porém, não se encontra na literatura a avaliação dos efeitos da representação ou abstração dos componentes do processo bioquímico descrito anteriormente na eficiência dos algoritmos evolucionários para otimização multiobjetivo, ou pelo menos não de forma satisfatória, ou seja, isolando cada elemento e utilizando um conjunto de problemas de testes conceituado e usando um indicador completo e compatível que agrega informações de proximidade da fronteira de Pareto e de sua distribuição.

1.2 Problemática e Solução Proposta

Realizar a exploração do espaço de projeto de sistemas embarcados usando algoritmos evolucionários não é uma ideia nova. Eckart Zitzler e Lothar Thiele já faziam com sucesso essa exploração em 1999 [Zitzler e Thiele 1999]. Os processos e recursos eram modelados como simples vértices num grafo dirigido com anotações do tempo de execução exigido pelos processos e a exploração consistia na escolha/alocação de recursos (processadores e barramentos) para suportar a execução de processos, o mapeamento de processos a processadores e o escalonamento dos processos no tempo, como mostra a figura 1.2. Esse nível de abstração na modelagem e suas atividades principais da exploração (alocação, mapeamento e escalonamento) permanecem praticamente idênticas até

[3] a *evolução* é definida como a mudança da frequência relativa de um gene na população.

hoje [Haubelt e Teich 2003, Schlichter et al. 2006, Pilato et al. 2008] Essas abordagens servem bem para uma visão de alto nível das opções de exploração, ou para a exploração de alguns parâmetros. Entretanto, para a geração automática de sistemas embarcados completos, esse tipo de abordagem é insuficiente.

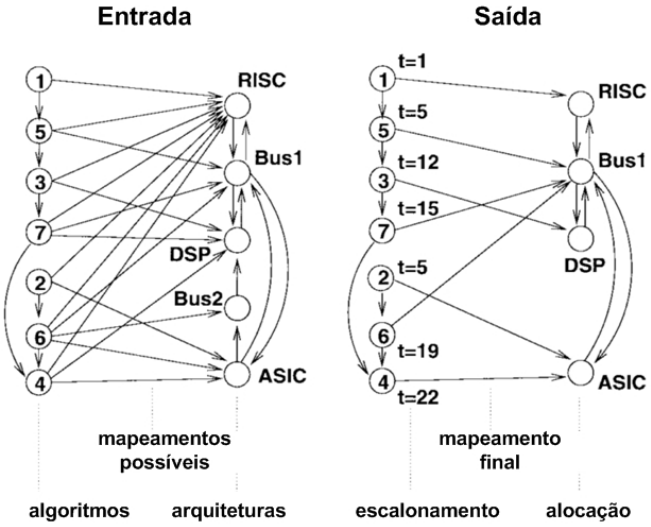


Figura 1.2: *Exploração do espaço de projeto em sistemas embarcados. Já em 1999 a exploração consistia, basicamente, na alocação de recursos de hardware, no mapeamento entre processos de software a recursos de hardware, e no escalonamento dos processos de software. Essa abordagem para exploração tem persistido na última década.*
Fonte: [Zitzler e Thiele 1999]

Exploração hierárquica do espaço de projeto em sistemas embarcados, visando a geração automática de sistemas funcionais é um desafio que persiste ainda atualmente, tanto em nível de software quanto de hardware. Do ponto de vista do software, o software embarcado e os processos de compilação/ligação associados são muito mais complexos que os modelos usualmente adotados para representá-los, limitando sua aplicação prática. Desenvolvimento de software baseado em componentes é a abordagem mais comum, e demanda que componentes de software sejam configurados e adaptados, exigindo exploração

de suas configurações (parâmetros) e de possíveis adaptações ao cenário em execução (normalmente associadas a requisitos não-funcionais, como sincronização e gerenciamento de energia). Há componentes alternativos que satisfazem o mesmo requisito (como duas implementações distintas do mesmo componente), dependências entre componentes (como um escalonador que depende de um *timer*) e exclusão mútua entre componentes (como gerenciamento de memória *flat* e segmentada), que precisam ser explorados e resolvidos. Há componentes independentes de arquitetura[4] e outros que executam apenas em arquiteturas específicas, limitando os mapeamentos possíveis entre software e processadores e complicando o processo de geração da imagem final do software. Há componentes descritos em diferentes linguagens e níveis de abstração (como C++, SystemC ou FSM), exigindo diferentes ferramentas de software (com diferentes parâmetros passíveis de exploração) para transformá-los em código executável numa arquitetura específica. Outros aspectos de sistema de suporte, como inicialização e mapa de memória também precisam ser tratados em diferentes arquiteturas. Uma DSE que considere todos esses aspectos ainda é um problema atual e relevante, e as soluções encontradas na literatura não representam todos os aspectos apresentados.

Do ponto de vista do hardware, a abordagem baseada em componentes também é a mais comum. O software embarcado pode executar numa arquitetura *softcore*[5] sintetizada juntamente com outros blocos sintetizáveis (*IP cores*) num dispositivo lógico programável (ex. FPGA), ou podem executar numa arquitetura física (ex. microcontrolador), soldada juntamente com outros componentes físicos numa placa de circuitos, ou então ambos, ou mesmo consistir de um novo dispositivo físico projetado especificamente para essa finalidade (ex. ASIC). Mesmo quando executando num *softcore* sintetizado num PLD (PLD - *Programmable Logic Device*), o próprio PLD faz parte de um suporte de hardware físico. As abordagens atuais geralmente fixam a plataforma-alvo (suporte de hardware), permitindo apenas sua parametrização, e ainda não consideram simultaneamente a exploração de suportes de hardware físico e sintetizáveis. Nesta tese, definimos um *suporte de hardware* como a ligação de componentes de hardware, de modo que formem um circuito eletrônico contendo pelo menos uma unidade de proces-

[4] Uma *arquitetura* é definida como a interface do hardware que é visível em nível de software e aos desenvolvedores do sistema, e que constitui-se, basicamente das instruções e registradores reconhecidos pela CPU e pelos mecanismos de acesso aos dispositivos.

[5] Uma *arquitetura softcore* é uma arquitetura implementada totalmente em lógica sintetizável, como linguagens de descrição de hardware.

samento e que forma a base sobre a qual executa o software. Entretanto, a própria decisão de usar PLD ou microcontroladores ou desenvolver um novo ASIC (ASIC - *Application Specific Integrated Circuit*) ou uma solução híbrida deveria ser resultado da exploração, e não pré-fixado. A utilização de suportes de hardware pré-definidos tem muitas vantagens, como a reutilização dos mesmos; entretanto, deveria ser possível explorar novos suportes de hardware, criados especificamente para um produto ou sistema embarcado. Projetar novo suporte de hardware (físico ou sintetizável) ou chips (ASICs) para cada produto é um enorme desafio. Entretanto, técnicas de evolução de hardware poderiam ajudar nessa exploração. Para que isso seja possível, é necessário representar componentes de hardware (físicos e sintetizáveis) com uma quantidade maior de detalhes do que a encontrada nas soluções atuais, incluindo informações lógicas/elétricas de cada pino e barramento. O projeto de ASICs exigiria ainda mais informações, como tecnologia de transistores, espaçamento e layout, entre muitas outras.

Um modelo de computação evolucionária que permita a exploração em sistemas embarcados dirigidos pela aplicação precisa de novas estruturas e operadores genéticos e epigenéticos. O genoma do indivíduo precisa representar as adaptações e configurações de cada componente, suas dependências de outros componentes, seu mapeamento a arquiteturas, a recursos de hardware, e seu escalonamento no tempo. Assim, várias informações relacionadas precisam ser representadas, principalmente o relacionamento entre genes (epistasia). Para viabilizar a exploração em vários níveis de abstração e também a possibilidade de '*templates*' de micro-arquiteturas ou de suporte de hardware, é necessário que a interação entre alguns genes e parte das dependências, mapeamentos e configurações seja mantida imutável (por decisão do projetista) enquanto outras devem ser alvo dos operadores de variação genética. A quantidade dos genes também não é constante, uma vez que o suporte de hardware pode evoluir de monoprocessado para multiprocessado, e com o aumento de componentes de hardware (representados por genes) também cresce a quantidade de componentes de software mapeados a eles. Técnicas adequadas para tratar soluções inviáveis devem ser usadas, pois dificilmente um indivíduo representa um sistema embarcado (software + hardware) totalmente viável. Nesse sentido também é importante a inclusão de elementos epigenéticos, de forma a alterar a estrutura genética com base em sua avaliação fenotípica, de modo, por exemplo, a marcar os genes que estão inviabilizando a solução e diminuir sua expressividade ou aumentar sua taxa de mutação. Por fim, a estrutura dos genes não é única, ou sua tradução fenotípica não é direta, pois representam entidades com características bem distintas.

Contudo, não está claro se a representação de novas estruturas e operadores nos algoritmos evolucionários é capaz de permitir a exploração do espaço de projeto de sistemas embarcados dirigidos pela aplicação em vários níveis de abstração e qual será a qualidade das soluções obtidas através desses novos algoritmos. Essas questões serão verificadas nesta tese.

Assim, esta tese trata do problema da exploração do espaço de projeto em diferentes níveis de abstração para sistemas embarcados dirigidos pela aplicação, através do desenvolvimento de metamodelos que representam o projeto de um sistema embarcado, os objetivos a serem otimizados na exploração, seus componentes de hardware e software, bem como o suporte de hardware sobre o qual executa, distinguindo o hardware físico de um possível hardware sintetizado num dispositivo lógico programável, como um FPGA (FPGA - *Field Programmable Gate Array*). A opção pelo projeto de novos chips (ASICs) como possível plataforma não foi considerada nesta tese por limitações no seu escopo. Na solução proposta esses metamodelos são mapeados num modelo evolucionário de otimização multiobjetivo específico para exploração do espaço de projeto de sistemas embarcados dirigidos pela aplicação. Nesse modelo evolucionário é avaliada a eficiência de cada adaptação realizada, em relação à otimização multiobjetivo. Entretanto, a avaliação da eficiência desses modelos em relação à exploração do espaço de projeto implica diversos outros problemas que extrapolam o escopo desta tese, como, por exemplo, a obtenção de estimativas precisas para cada objetivo a ser otimizado no sistema, a inexistência de expressões matemáticas das funções-objetivo, o desconhecimento da fronteira de Pareto, a implementação dos componentes de software e de hardware sintetizável que compõem o sistema e a montagem do suporte de hardware, e a obtenção de uma amostra razoável de sistemas gerados, o que exige uma limitação dos objetivos a serem alcançados e do escopo do problema tratado, apresentados a seguir.

1.3 Objetivos

O principal objetivo desta tese é o desenvolvimento de um modelo evolucionário que permita a exploração de espaço de projeto de sistemas embarcados dirigidos pela aplicação em diferentes níveis de abstração. Tal modelo evolucionário deve suportar a metodologia de desenvolvimento de sistemas embarcados dirigidos pela aplicação (ADESD) e seu fluxo de projeto, representando ele-

mentos normalmente abstraídos em outras soluções, como a configuração e adaptação de componentes de software e a evolução do próprio suporte de hardware (físico ou sintetizável).

Neste contexto, são objetivos específicos desta tese avaliar as novas estruturas e operadores de inspiração biológica desenvolvidos para o modelo evolucionário de exploração do espaço de projeto, e que incluem:

- Auto-adaptação de parâmetros de operadores genéticos;
- Elementos epigenéticos;
- Neutralidade genética;
- Diploidismo e competição genética por transcrição.

1.4 Escopo do Problema e Limitações

Nesta tese desenvolveu-se um modelo evolucionário para otimização multiobjetivo, que faz parte de um *framework* de otimização, concebido para esse fim e integrado à suite de ferramentas da ADESD para geração de sistemas embarcados. A metodologia ADESD foi escolhida por suas características em relação ao projeto de sistemas embarcados e também por ser utilizada no grupo de pesquisa deste autor. O modelo evolucionário para otimização multiobjetivo é baseado nas estruturas propostas pela programação genética (GP - *Genetic Programming*) e pela programação da expressão genética (GEP - *Gene Expression Programming*). Porém, como citado nas seções anteriores, novas características foram incluídas. Fazem parte do escopo desta pesquisa a representação de elementos básicos de informação dentro dos genes, a inclusão de uma estrutura epigenética para cada componente genético (códon, gene e cromossomo), que regula sua expressão, determina o comportamento de operadores para esse componente, entre outras funções específicas para cada componente genético. Também foram incluídos novos tipos de genes, mais adequados à representação de algumas configurações de componentes de sistemas embarcados e seu mapeamento a elementos arquiteturais. Foram representados os *loci*[6] nos cromossomos, permitindo que genes alelos compitam por sua expressão e participação no fenótipo do indivíduo, ou seja,

[6]loci é o plural de locus, que é a posição específica de um gene num cromossomo.

componentes competindo para fazer parte do sistema embarcado. Essas novas estruturas exigiram o desenvolvimento de novas versões de operadores genéticos como mutação, recombinação, adição e deleção. Contudo, não foram representados níveis adicionais de transformação entre genótipo e fenótipo, como RNAs (RNA - *Ribonucleic Acid*), interações entre proteínas e ciclos metabólicos. Essas transformações são muito dependentes do problema sendo tratado e não foram desenvolvidas, embora o arcabouço para seu desenvolvimento o tenha sido.

O modelo para otimização supracitado corresponde ao mapeamento dos outros modelos e metamodelos desenvolvidos nesta tese para uma estrutura baseada em algoritmos evolucionários que permita a otimização multiobjetivo e, desse modo, a exploração do espaço de projeto. Esses outros modelos, que formam a base para o modelo de otimização, representam um sistema embarcado, conforme a metodologia ADESD, objetivos que podem ser otimizados e a evolução de suas equações, e componentes físicos (eletro-eletrônicos, mecânicos) e *cyber*/lógicos (software, hardware sintetizável) que formam um repositório de componentes usado para compor tais sistemas embarcados. Faz parte do escopo do problema tratado a especificação de restrições de projeto e de um conjunto de funções-objetivo a serem minimizadas. O modelo suporta as características da ADESD, o que inclui componentes configuráveis e adaptáveis, e a adaptação do próprio suporte de hardware, que é composto por componentes físicos, possivelmente incluídos neles dispositivos lógicos programáveis, dentro dos quais são sintetizados hardwares descritos numa linguagem de descrição de hardware. Porém, a possibilidade de projetar um novo ASIC como resultado da exploração não foi considerada por limitações no escopo do projeto. Apesar do modelo ser possivelmente expansível para exploração do espaço de projeto em níveis lógico e físico, permitindo otimização de *place and rounting* e *layout* de transistores no substrato de silício de um chip, essa alternativa não faz parte do escopo desta tese, cujo elemento de mais baixo nível considerado é um componente eletrônico prémanufaturado, como microcontroladores, FPGAs, controladores de interrupção, e outros. Também consideramos como componentes de mais alto nível os componentes de software já implementados.

Embora os modelos desenvolvidos, devidamente manipulados pela suite de ferramentas de apoio da ADESD, permitam o desenvolvimento de sistemas embarcados reais, esses sistemas reais não são avaliados nesta tese. As equações que descrevem os objetivos a serem minimizados num sistema embarcado (consumo de energia, tempo de execução, etc) são desconhecidas e, portanto, também é desconhecida sua fronteira de Pareto. Portanto, é impraticável demonstrar que a

exploração escolheu, efetivamente, as melhores soluções para um projeto de sistema embarcado qualquer. Outras pesquisas de exploração do espaço de projeto encontradas na literatura usaram outros componentes de software e de hardware, com outras implementações, possivelmente outras linguagens, com outros compiladores e sintetizadores, ou componentes apenas descritos em modelos simbólicos e gerando apenas resultados simulados. Desse modo, os sistemas embarcados gerados com o *framework* desenvolvido e os componentes existentes no repositório da ADESD não são diretamente comparáveis com as pesquisas da literatura.

Uma boa amostra de sistemas embarcados gerados talvez pudesse demonstrar a eficiência da exploração do espaço de projeto realizada com os modelos desenvolvidos nesta tese. Porém, isso exigiria a obtenção de valores reais de consumo de energia, tempo de execução, área de silício ocupada, entre outros, para todos os sistemas gerados, o que demanda também o desenvolvimento (ou pelo menos a integração) de co-simuladores e outros estimadores. Além disso, a abordagem utilizada propõe a exploração do suporte de hardware, mas ela não é capaz de gerar automaticamente circuitos eletrônicos completamente funcionais. A exploração do suporte de hardware significa que são geradas diferentes versões de circuitos de hardware físico e/ou sintetizável, que consistem de diferentes componentes eletrônicos (ou *IP cores* sintetizáveis) interconectados, respeitando vários tipos de restrições, como largura de dados, tensões e correntes compatíveis, entre outras. Contudo, mesmo com uma boa representação de características elétricas e dos pinos e barramentos desses componentes, o modelo é capaz de fornecer apenas um esboço inicial do esquema eletrônico do circuito, mas não é capaz de inferir a existência de cada resistor, cada capacitor e demais componentes eletrônicos, e ainda dimensionar adequadamente cada um deles, gerando um circuito eletrônico completo e totalmente funcional. Essa exploração também não produz o layout de placas de circuito impresso para os circuitos físicos gerados.

Portanto, mesmo provendo uma boa estimativa do que seria um suporte de hardware adequado, ainda exige-se do projetista muito esforço e bons conhecimentos em eletrônica para que o hardware torne-se funcional. Por esses motivos, não é viável uma avaliação direta dos resultados da exploração do espaço de projeto em sistemas embarcados reais gerados como resultados desta tese. Entretanto, dado que a exploração do espaço de projeto consiste de um problema de otimização multiobjetivo, e que os conjuntos de problemas de teste (*benchmarks* de otimização) foram criados para impor diferentes dificuldades aos modelos de otimização, que são encontradas em problemas reais (como o projeto de

sistemas embarcados), então a avaliação da qualidade da otimização em problemas de testes cuja fronteira de Pareto seja conhecida, como foi realizado nesta tese, parece ser uma boa estimativa para a eficiência da exploração do espaço de projeto em sistemas embarcados reais.

1.5 Visão Geral da Tese

Terminada a parte introdutória com este capítulo, a segunda parte desta tese apresenta a fundamentação e o estado-da-arte dos principais temas relacionados, quais sejam, a otimização multiobjetivo, os algoritmos evolucionários e a exploração do espaço de projeto em sistemas embarcados, e é composta pelos capítulos 2, 3 e 4.

O capítulo 2 trata da otimização multiobjetivo, e inicia mostrando que otimização multiobjetivo visa escolher as melhores soluções conforme as preferências do projetista, e não apenas encontrar o conjunto de Pareto, como tem sido sugerido em muitas pesquisas. Certamente, uma forma muito útil e difundida de representar o que são as melhores soluções é o conjunto de Pareto. Entretanto, tento diferenciar o processo de busca de um conjunto de soluções das preferências do projetista, e também elucidar as limitações da dominância de Pareto em problemas envolvendo muitos objetivos, que pode ser o caso dos sistemas embarcados. Nesta tese proponho verificar se a representação de alguns processos bioquímicos associados à genética devem ou não ser modelados computacionalmente em algoritmos evolucionários, pelo menos para problemas em que não se obtém facilmente um modelo matemático, que também pode ser o caso dos sistemas embarcados. Para tanto, faz-se necessário utilizar indicadores de qualidade confiáveis e que avaliem as características de um bom conjunto de soluções produzidas por um otimizador. Proponho o uso do indicador hipervolume, que tem ganho grande destaque nos últimos anos, e o apresento nesse capítulo. Para permitir a comparação com heurísticas já existentes, também faz-se necessário um conjunto de problemas de testes adequado. Então são apresentadas quais as características desejadas de um conjunto de problemas de testes adequado. Vários conjuntos de teste foram propostos, mas a maioria deles possui limitações, como a quantidade de funções-objetivo utilizadas. Por outro lado, a criação de um novo conjunto de problemas de testes pode ser complexa. Assim, o capítulo termina apresentando conjuntos de testes encontrados na literatura e que são consider-

ados adequados pela comunidade acadêmica. Por fim, e apenas para contextualizar o leitor e para justificar a escolha de algoritmos evolucionários, apresento as características gerais de técnicas determinísticas e estocásticas de otimização multiobjetivo, incluindo os algoritmos evolucionários.

A ideia principal do capítulo 3 é apresentar as principais questões associadas aos algoritmos evolucionários usados para otimização multiobjetivo para propor a incorporação de elementos normalmente neles abstraídos e que podem ser úteis ao projeto de sistemas embarcados. O capítulo inicia com uma descrição do processo bioquímico associado à genética, e que suporta a evolução das espécies. Pelo menos nos seres vivos (e não nos modelos computacionais) tento mostrar a importância de regiões não codificantes, da regulação da expressão genética, da interação e da competição entre genes, e da auto-adaptação, que foram incorporados no modelo de otimização desenvolvido nesta tese, e também a importância dos diferentes níveis de decodificação e de transformação do genoma até o fenótipo, mas que não foram avaliados nesta pesquisa. As diferentes abordagens associadas à seleção de indivíduos, elitismo, diversidade da população e parametrização são apresentadas em seguida. Ênfase é dada às abordagens mais difundidas e que são utilizadas nesta tese, mesmo que sejam utilizadas apenas para comparação de sua eficiência, mensurada usando o hipervolume, em relação aos elementos que incorporei no modelo evolucionário desenvolvido.

O capítulo 4 inicia com o projeto de sistemas embarcados, mostrando que diferentes fluxos de projeto foram propostos, mas que são distintos das metodologias que os usam. Duas metodologias de projeto são apresentadas: O desenvolvimento baseado em plataforma (PbD), amplamente difundido, e o desenvolvimento de sistemas embarcados dirigidos pela aplicação (ADESD). Suas características e conceitos principais são apresentados, o que nos leva a um dos objetivos desta tese, que é a criação de um modelo que permita a exploração do espaço de projeto em sistemas projetados com a metodologia ADESD. A suite de ferramentas da ADESD também é apresentada, já que os modelos e *framework* de otimização desenvolvidos foram integrados a ela. O capítulo passa então a focar na etapa de exploração do espaço de projeto e apresenta pesquisas correlatas que fizeram uso de diferentes técnicas de otimização e também metodologias de projeto. Mostro as vantagens e limitações dessas pesquisas e que me guiaram às adaptações propostas no modelo de exploração desenvolvido.

A terceira parte desta tese trata do desenvolvimento da solução proposta, e da apresentação e análise dos resultados obtidos. Ela é dividida em 3 capítulos,

que descrevem os modelos e metamodelos que representam sistemas embarcados (capítulo 5), o modelo evolucionário para exploração do espaço de projeto (capítulo 6) e a avaliação desse modelo (capítulo 7).

O capítulo 5 inicia relembrando algumas características da metodologia ADESD para o projeto de sistemas embarcados e como o projeto de um sistema embarcado é modelado nesta tese, criando o que foi denominado "metamodelo de projeto de sistema embarcado". Tento enfatizar a representação de restrições de projeto e de como as entradas e saídas de cada projeto realizado podem ser usadas para auxiliar no projeto de futuros sistemas.O capítulo prossegue descrevendo o "metamodelo de componentes". Destaco como esse metamodelo representa as características necessárias ao projeto com a ADESD, como o agrupamento de componentes em famílias, as interfaces infladas, a aplicação de aspectos, as funcionalidades configuráveis, as dependências, configurações e mapeamentos, entre outras. Também é destacada nessa parte do capítulo a aplicabilidade do metamodelo na representação de componentes híbridos de software e hardware. Outra contribuição desse modelo é a distinção entre componentes de hardware físico de componentes de hardware sintetizável, e o mapeamento de componentes de hardware sintetizável a um componente de hardware físico programável (PLD). Mostra-se como essa característica permite explorar o suporte de hardware em dois níveis diferentes: o nível físico, composto de componentes eletrônicos numa placa de circuito impresso e, quando um desses componentes físicos é um dispositivo lógico programável (PLD), também o nível lógico, com o suporte de hardware sintetizável dentro desse PLD. O capítulo prossegue apresentando como esses metamodelos foram incorporados na suite de ferramentas da ADESD para suportar a exploração do espaço de projeto. O projeto de um sistema embarcado deve ser mapeado a um modelo evolucionário e ser utilizado por um otimizador para que a exploração ocorra. Assim, o capítulo descreve o *framework* de otimização que foi desenvolvido nesta tese, que apresenta características encontradas em outros *frameworks* de otimização, como o PISA e o OPT4J, e que foi efetivamente baseado neles.

O capítulo 6 apresenta o modelo evolucionário de otimização criado para posteriormente suportar a exploração do espaço de projeto. Nesse capítulo são apresentados aspectos de inspiração biológica, principalmente elementos epigenéticos, neutralidade genética, multiploidismo e competição genética. Esses novos aspectos podem ou não ser incorporados aos algoritmos evolucionários, conforme desejo do projetista. Com isso é possível avaliar a eficiência individual de cada adaptação incluída. O capítulo prossegue ilustrando um exemplo de como

essas adaptações podem ser usadas para representação de um sistema embarcado projetado pela ADESD através de um modelo evolucionário e, com isso, realizar sua otimização. O capítulo termina mostrando o modelo de um indivíduo, que representa uma solução do problema sendo tratado, e foca em seu genótipo, que tem sua estrutura alterada pelas adaptações criadas.

A avaliação da qualidade de otimização do modelo evolucionário de otimização é apresentada no capítulo 7. O capítulo começa explicitando que informações se deseja extrair dos experimentos, e que correspondem a parte das contribuições desta tese. Essas informações incluem quais elementos do processo bioquímico que foram incluídos no modelo de otimização trazem algum benefício ao processo de otimização. Experimentos para obter essas informações são planejados e descritos. Eles incluem, basicamente, a otimização do conjunto de funções de teste ZDT e sua avaliação usando o indicador hipervolume. Para cada função de teste são variadas as heurísticas do modelo de otimização que serão comparadas. O capítulo prossegue apresentando os resultados em diferentes figuras e tabelas, destacando os efeitos de cada diferente aspecto desenvolvido. Por fim, discuto os resultados da avaliação.

A parte final desta tese é composta pelo capítulo 8, último capítulo desta tese. Esse último capítulo apresenta as conclusões finais, ressaltando as contribuições desta tese, suas vantagens e limitações, e também indicando possibilidades de futuras pesquisas relacionadas com os desenvolvimentos feitos. A parte pós-textual é composta por alguns apêndices, que apresentam alguns aspectos associados aos desenvolvimentos feitos nesta tese, mas que não foram incluídos à parte textual pois não foi feita uma avaliação formal sobre eles. Em especial, é apresentada uma breve descrição da nova suite de ferramentas da ADESD à qual foram incorporados todos os modelos e metamodelos apresentados, e também uma relação dos componentes que foram cadastrados no repositório de componentes dessa suite de ferramentas.

PARTE II

Fundamentação e Estado-da-Arte

CAPÍTULO 2

OTIMIZAÇÃO MULTIOBJETIVO

Da guerra da natureza, da fome e da morte, advém diretamente o mais elevado objetivo que podemos conceber, nomeadamente a produção de animais mais perfeitos.
—CHARLES DARWIN (The Descent of Man, 1871)

Este capítulo sobre otimização multiobjetivo é dividido em 4 seções. A seção 2.1 apresenta a formulação matemática do problema de otimização e dos espaços envolvidos, bem como o conceito de soluções preferíveis, que guia otimizadores na seleção das melhores soluções. A seção 2.2 apresenta a dominância de Pareto, amplamente difundida, alternativas a essa dominância, e também formas de encontrar uma solução dentre o conjunto de soluções encontradas, conforme preferências do projetista. A seção 2.3 apresenta alguns indicadores de qualidade para heurísticas de otimização que permitam sua comparação e avaliação quantitativa, focando no indicador de hipervolume. Essa seção apresenta também as características desejáveis em conjuntos de funções de teste e que tipos de obstáculos eles podem prover aos otimizadores sob teste. Por fim, a seção 2.4 apresenta as diferentes técnicas clássicas e estocásticas para a otimização multiobjetivo, como motivação à utilização dos algoritmos evolucionários, alvo do capítulo 3.

2.1 Definições

Problemas de otimização simples visam encontrar uma solução ótima que maximiza ou minimiza uma única *função-objetivo*. Dado que um problema de maximização pode ser transformado num problema de minimização, pode-se afirmar, sem perda de generalidade, que um problema de otimização visa minimizar uma função-objetivo [Benedetti, Farina e Gobbi 2006]. Todavia, problemas de otimização envolvendo mais de uma função são comuns no mundo real. Um problema de otimização multiobjetivo (MOP - *Multiobjective Problem*) visa minimizar um conjunto de funções-objetivo simultaneamente [Coello 2006]. Formalmente, um MOP visa:

$$\text{minimizar } \mathscr{F}(\overrightarrow{x}) = \{f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), ..., f_M(\overrightarrow{x})\} \tag{2.1}$$

sujeito a:

$$r(\overrightarrow{x}) = \{r_1(\overrightarrow{x}), r_2(\overrightarrow{x}), ..., r_s(\overrightarrow{x})\}\ , r_i(\overrightarrow{x}) \leq 0\ ,\ \forall i = 1,2,...,s \qquad (2.2)$$

$$r_x(\overrightarrow{x}) = \{r_{x1}(\overrightarrow{x}), r_{x2}(\overrightarrow{x}), ..., r_{xt}(\overrightarrow{x})\}\ , r_{xi}(\overrightarrow{x}) \leq 1\ ,\ \forall i = 1,2,...,t \qquad (2.3)$$

onde $\overrightarrow{x} = [\overrightarrow{x_1}, \overrightarrow{x_2}, ..., \overrightarrow{x_v}] \in \mathscr{X}$ é o vetor das variáveis de decisão do problema, $\mathscr{F}$ é o conjunto de funções-objetivo a ser minimizado, $f_i : \Re^v \mapsto \Re$ é uma função-objetivo que mapeia um vetor de decisão $\overrightarrow{x_i}$ a um valor real que quantifica esse vetor de decisão sobre certo aspecto do problema (como energia consumida e tempo de execução, por exemplo), $r_i : \Re^s \mapsto \Re$ é uma função de restrição do problema e que representa, tipicamente, recursos limitados ou requisitos de projeto (ex: $area + 50 \leq 0$). Já uma restrição mutuamente exclusiva $r_{xi} : \aleph^s \mapsto \aleph^+$ assume a forma $r_{xi}(\overrightarrow{x_i}) = x_1 + x_2 + ... + x_n$, onde $x_1, x_2, ..., x_n \in \overrightarrow{x_i}$ são variáveis de decisão binárias (0/1) e que representam diferentes alternativas para certa escolha de projeto, e na qual apenas uma alternativa pode ser selecionada (ex: particionamento hardware/software). Esse tipo de restrição é muito comum no projeto de sistemas embarcados [Chakraborty, Mitra e Roychoudhury 2006]. Diz-se que uma *solução viável* é uma solução que não viola nenhuma das restrições impostas, definida formalmente na equação (2.4)

$$\overrightarrow{x} \in \mathscr{X}_{VIAVEL} : r_i(\overrightarrow{x}) \leq 0, \forall i = 1,2,...,s \wedge r_{xj}(\overrightarrow{x}) \leq 1, \forall j = 1,2,...,t \qquad (2.4)$$

Um MOP pode ser então definido como uma quádrupla $(\mathscr{X}, \mathscr{Z}, \mathscr{F}, r)$, onde $\mathscr{X} = \{\overrightarrow{x_1}, ..., \overrightarrow{x_n}\}$ é o *espaço de decisões* e seus elementos $\overrightarrow{x_i}$ são *vetores de decisão* ou simplesmente possíveis *soluções* ao problema. $\mathscr{Z} : \Re^M$ é o *espaço de soluções*, em que os *vetores de decisão* são avaliados e comparados entre si. Os elementos de $\mathscr{Z}$ são *vetores-objetivo*. Idealmente, um MOP deve minimizar todos os elementos desse vetor. $\mathscr{F}$ é o conjunto de *funções-objetivo* (eq. (2.1)), $f_i : \mathscr{X} \mapsto \mathscr{Z}$, que mapeia *vetores de decisão* no *espaço de decisões* a um ponto (*M-dimensional*) no *espaço de soluções* [Zitzler e Thiele 1999]. Uma representação desses elementos pode ser vista na figura 2.1. De maneira geral, um algoritmo ou técnica de resolução de um MOP gera uma solução $\overrightarrow{x}$ no *espaço de soluções* e, usando as *funções-objetivo*, gera um *vetor-objetivo* $\overrightarrow{z} = \{z_1, z_2, ..., z_M\}$, $z_i = f_i(\overrightarrow{x})$, $i = 1,2,...,M$ que avalia a qualidade dessa solução em relação a cada objetivo. Esse algoritmo pode utilizar os *vetores-objetivo* já conhecidos para selecionar novas soluções que possivelmente correspondem a melhores *vetores-objetivo*, de forma a minimizá-lo. Assim, quando devidamente formulado, a ad-

equação de um projeto é representada unicamente por sua localização no espaço de soluções [Di Nuovo et al. 2006].

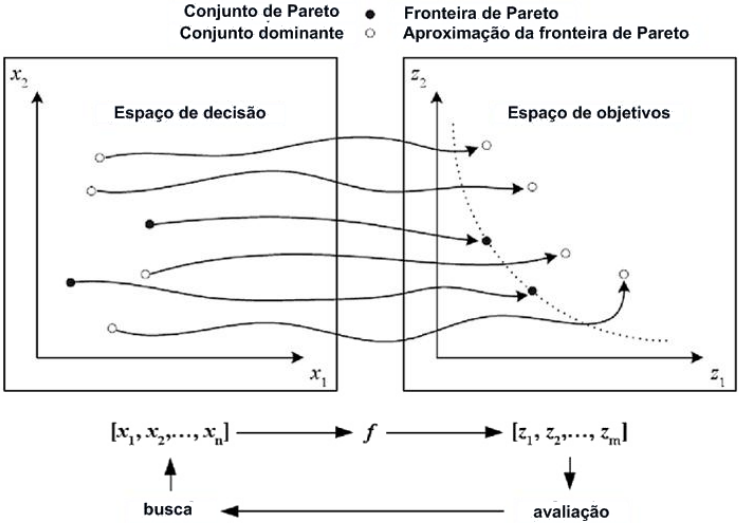
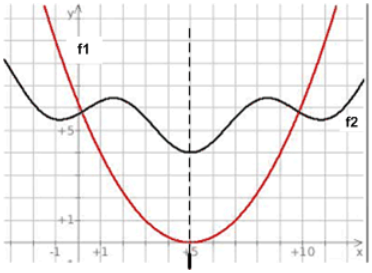


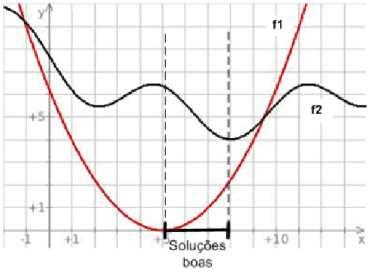
Figura 2.1: *Espaços de decisão e de objetivos num problema de otimização multiobjetivo. Vetores de decisão no espaço de decisões são avaliados e mapeados ao espaço de soluções, em que podem ser comparados em busca de soluções ótimas.*
Fonte: [Zitzler e Thiele 1999]

Por fim, $r \subseteq \mathscr{Z} x \mathscr{Z}$ é uma relação sobre o *espaço de soluções* e representa um ordenamento parcial sobre $\mathscr{Z} : \Re^M$, permitindo a escolha das melhores soluções, necessária pois as *funções-objetivo* f_i são normalmente conflitantes entre si, o que torna praticamente impossível a existência de uma única solução ótima, formalmente definida na equação (2.5). A figura 1.1, apresentada na seção 1.1, e reapresentada abaixo na figura 2.2 por conveniência, ilustra essas condições: na figura 2.2(a) é apresentado o caso incomum, em que ambas funções possuem exatamente o mesmo ponto mínimo, de forma que existe uma solução ótima. Esse caso torna-se praticamente inexistente com o aumento da quantidade de funções-objetivo [Said, Bechikh e Ghédira 2010, Lara et al. 2010]. A figura 2.2(b) mostra o caso mais comum, em que não existe uma solução que minimiza simultaneamente todas as funções-objetivo. Nesses casos, deve-se buscar soluções que são preferíveis em relação a outras soluções por algum critério bem definido.

$$\overrightarrow{x}_{OTM} \in \mathscr{X} : \forall \overrightarrow{x} \in \mathscr{X},\ f_i(\overrightarrow{x}_{OTM}) \leq f_i(\overrightarrow{x}),\ 1 \leq i \leq M \tag{2.5}$$



(a) Existência de uma solução ótima



(b) Inexistência de uma solução ótima

Figura 2.2: *Exemplo da existência ou não de uma solução ótima. (REAPRESENTAÇÃO DA FIGURA 1.1). Existência ou inexistência de uma solução ótima que minimiza simultaneamente ambas as funções-objetivo de um problema de otimização multiobjetivo. Em geral, as funções-objetivo são conflitantes, de modo que não existe uma solução ótima que as minimiza simultaneamente*

As diversas funções-objetivo f_i usualmente estão em diferentes escalas e a normalização de todas as funções-objetivo pode ser usada para mapear todos os valores para o intervalo $\Re[0,1]$, gerando um problema de otimização não dimensionalizado (sem unidades). Experimentos demonstraram que as escalas dos objetivos têm efeitos sobre os resultados, inclusive se os mesmos algoritmos forem usados [Zou et al. 2008]. Uma função-objetivo f_i pode ser normalizada para uma função $\hat{f}_i$ conforme é apresentado na equação (2.6), o que demanda o conhecimento dos valores mínimo ($f_{i,min}$) e máximo ($f_{i,max}$) de cada função-objetivo. Se esses valores não forem conhecidos, é possível estimá-los usando técnicas de otimização de um único objetivo, já que não é imprescindível ter-se os valores exatos [Zou et al. 2008].

$$\hat{f}_i(\overrightarrow{x}) = \frac{f_i(\overrightarrow{x}) - f_{i,min}}{f_{i,max} - f_{i,min}} \tag{2.6}$$

2.2 Soluções Preferíveis

Funções-objetivo conflitantes num MOP implicam a inexistência de uma solução ótima única [Said, Bechikh e Ghédira 2010]. As técnicas de resolução de MOPs buscam encontrar um conjunto de boas soluções que sejam preferíveis em relação a outras soluções não tão boas. É necessário então definir relações de preferência de soluções. Podemos definir a relação $\vec{x} \preceq \vec{y}$ sobre as soluções $\vec{x}$ e $\vec{y}$ do espaço de decisões ($\vec{x}, \vec{y} \in \mathscr{X}$) significando que a solução $\vec{x}$ não é pior que a solução $\vec{y}$, ou seja, é pelo menos *tão boa quanto* a solução $\vec{y}$ e é, portanto, fracamente preferível. Essa relação é transitiva, dado que $\vec{x} \preceq \vec{y} \wedge \vec{y} \preceq \vec{z} \Rightarrow \vec{x} \preceq \vec{z}$, e também é reflexiva, dado que $\vec{x} \preceq \vec{x}$.

Podemos também definir a relação $\vec{x} \prec \vec{y}$, significando que a solução $\vec{x}$ é melhor que a solução $\vec{y}$ em pelo menos um objetivo e é, portanto, preferível. Formalmente essa relação é definida na equação (2.7). Podemos ter uma preferência ainda maior num conjunto de soluções definindo a relação $\vec{x} \preccurlyeq \vec{y}$, significando que a solução $\vec{x}$ é melhor que a solução $\vec{y}$ em todos os objetivos e é, portanto, fortemente preferível. Essa relação pode ser formalmente definida na equação (2.8). Duas soluções podem ainda ser incomparáveis entre si, sem que uma seja preferível à outra, representada pela relação reflexiva e simétrica de incomparabilidade $\vec{x} \parallel \vec{y}$ e formalmente definida na equação (2.9), ou então duas soluções podem ser equivalentes, representada pela relação simétrica de equivalência $\vec{x} \backsimeq \vec{y}$ e formalmente definida na equação (2.10).

$$\vec{x} \prec \vec{y} \quad : \quad \vec{x} \preceq \vec{y} \wedge \vec{y} \npreceq \vec{x} \tag{2.7}$$

$$\vec{x} \preccurlyeq \vec{y} \quad : \quad \vec{x} \prec \vec{y} \wedge \vec{y} \npreceq \vec{x} \tag{2.8}$$

$$\vec{x} \parallel \vec{y} \quad : \quad \vec{x} \npreceq \vec{y} \wedge \vec{y} \npreceq \vec{x} \tag{2.9}$$

$$\vec{x} \backsimeq \vec{y} \quad : \quad \vec{x} \preceq \vec{y} \wedge \vec{y} \preceq \vec{x} \tag{2.10}$$

Buscar um conjunto de soluções preferíveis, então, depende do que significa "*tão boa quanto*" para poder formalizar a relação '$\preceq$'. Encontrar esse conjunto de soluções pode ser complexo e até impossível, pois o conjunto pode ser infinito. Assim, normalmente limita-se o algoritmo de otimização à localização de um sub-conjunto de soluções preferíveis ou a uma aproximação para tal conjunto. Para o projetista, entretanto, obter dezenas ou centenas de soluções pertencentes a esse conjunto não é de grande utilidade, uma vez que geralmente busca-se

uma única solução. Assim, outras técnicas devem ser usadas para a escolha de uma solução entre todas as encontradas. Uma formalização de "preferível" que guia a busca por um conjunto de soluções bastante difundida é a dominância de Pareto, descrita na seção seguinte, embora existam alternativas, apresentadas na seção 2.2.2. Técnicas para a escolha de uma solução final são apresentadas na seção 2.2.3.

2.2.1 Dominância de Pareto

Uma das formas mais conhecidas para especificar preferências entre soluções é a dominância de Edgenworth-Pareto, ou simplesmente dominância de Pareto. A dominância de Pareto é utilizada para selecionar um conjunto de soluções, o conjunto de Pareto, que representa diferentes compromissos (*trade-offs*) entre as *funções-objetivo*. Uma solução é preferível, ou seja, pertence ao conjunto de Pareto, se não há outra solução que possa melhorar pelo menos uma das *funções-objetivo* sem piorar outra; neste caso, diz-se que essa solução é Pareto-dominante [Zitzler e Thiele 1999], ou que ela *domina* outras soluções. Assim, define-se a relação de dominância fraca de Pareto $\overrightarrow{x} \preceq_p \overrightarrow{y}$, significando que a solução $\overrightarrow{x}$ é pelo menos tão boa quanto a solução $\overrightarrow{y}$, apresentada formalmente na equação (2.11).

$$\overrightarrow{x} \preceq_p \overrightarrow{y} : f_i(\overrightarrow{x}) \leq f_i(\overrightarrow{y}) \, , \, \forall f_i \in \mathscr{F} \tag{2.11}$$

Pode-se então definir um *conjunto de Pareto* $\mathscr{P}^*$ como sendo o conjunto de soluções que não são *dominadas* por nenhuma outra solução do *espaço de soluções*, formalmente apresentado na equação (2.12). Por fim, a *fronteira de Pareto* $\mathscr{PF}^*$ é a imagem do conjunto de Pareto sobre $\Re^M$, normalmente usado para visualização gráfica e comparação dos *trade-offs* entre as *soluções*, formalmente definido na equação (2.13). Uma representação da fronteira de Pareto para duas funções-objetivo pode ser vista na figura 2.3. Nessa figura, a linha contínua representa a fronteira de Pareto, e é demarcada pelas soluções não-dominadas (círculos preenchidos), dentre as quais a figura destaca a solução x_1. A área hachurada, acima e à direita de x_1, denota a área dominada por essa solução, ou seja, a porção do espaço de decisão no qual outras soluções são consideradas piores, não preferíveis, ou dominadas em relação a x_1, como as soluções x_2, x_3, x_4 e x_5, por exemplo. Dessas, x_5 também é uma solução que viola uma restrição

de projeto, que limita a área máxima do sistema, que é representada pela linha tracejada correspondente. Tratando-se de um problema de minimização, e sendo a fronteira de Pareto o melhor compromisso entre os objetivos, todas as demais soluções viáveis (círculos vazados) ficam acima e à direita da fronteira. Enquanto o conjunto de Pareto permite identificar o tipo de dificuldades encontradas no espaço de buscas, a fronteira de Pareto representa um subconjunto representativo do conjunto de Pareto [Huband et al. 2006].

$$\mathscr{P}^* = \{\overrightarrow{x} \in \mathscr{X} \mid \nexists \overrightarrow{y} \in \mathscr{X}, \overrightarrow{y} \preceq_p \overrightarrow{x}\} \quad (2.12)$$

$$\mathscr{PF}^* = \{\mathscr{F}(\overrightarrow{x}) \in \Re^M \mid \overrightarrow{x} \in \mathscr{P}^*\} \quad (2.13)$$

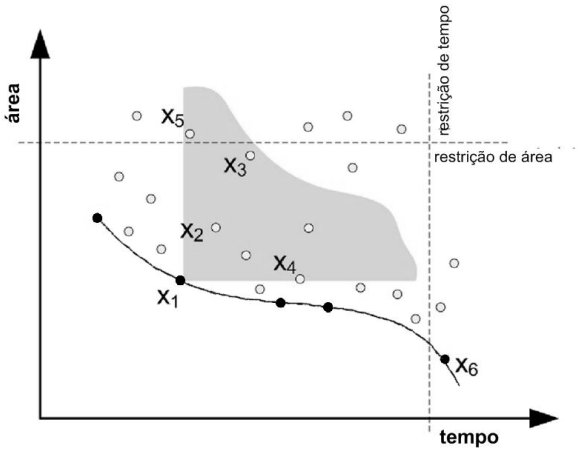


Figura 2.3: *Representação de soluções na fronteira de Pareto. Todas as soluções dominantes são representadas pelos círculos preenchidos, e correspondem às soluções no qual não existe outra solução que melhore ambos objetivos simultaneamente. Fonte: [Zitzler e Thiele 1999]*

Porém, encontrar o conjunto de Pareto ainda é complexo ou impossível. Um dos problemas que já foi citado é que esse conjunto pode ser infinito. Outro problema está relacionado à forma da fronteira de Pareto, que

pode ser complexa, além das soluções não-dominadas estarem mal-distribuídas. Um último problema ocorre com o aumento da quantidade de objetivos. Para problemas com mais de dois ou três objetivos, a dominância de Pareto não funciona bem, pois muitas soluções passam a ser dominantes [Fonseca 1995, Zou et al. 2008, Ishibuchi, Tsukamoto e Nojima 2008, Ishibuchi et al. 2009]. Se muitos indivíduos da população atual for em dominantes, não há pressão de seleção para que estratégias baseadas nessa dominância façam a população convergir em direção à fronteira de Pareto. Além disso, as soluções preferíveis, pela definição de Pareto, não consideram 3 aspectos que podem ser importantes [Benedetti, Farina e Gobbi 2006]: (1) A quantidade de objetivos melhorados ou piorados; (2) A relevância de tais melhoras ou pioras; e (3) A preferência do projetista entre objetivos. Dessa forma, alternativas à dominância de Pareto devem ser investigadas, e Coello [Coello 2009] afirma que métodos mais flexíveis de dominância ainda são temas relevantes de pesquisa.

2.2.2 Dominâncias Alternativas

Visando suprir as deficiências da dominância de Pareto na identificação de soluções preferíveis, várias alternativas foram propostas nos últimos anos. Kokolo et alli [Ikeda, Kita e Kobayashi 2001] propuseram a α-dominância, que define limites superiores e inferiores para os compromissos (*trade-offs*) entre os objetivos, de modo que uma pequena piora num dos objetivo é permitida, desde que cause uma melhora considerável em outros objetivos.

Laumanns et alli propuseram a ε-dominância [Laumanns et al. 2002], em que $\varepsilon \in \Re$ representa uma tolerância. Diz-se que uma solução $\overrightarrow{x} \in \overrightarrow{X}$ é ε-dominante em relação a outra solução $\overrightarrow{y} \in \overrightarrow{X}$ se ela atende à condição da equação (2.14), e usa-se a notação $\overrightarrow{x} \preceq_\varepsilon \overrightarrow{y}$ para representá-la.

$$\overrightarrow{x} \preceq_\varepsilon \overrightarrow{y} : \forall f_i \in \mathscr{F} \rightarrow (1+\varepsilon) f_i(\overrightarrow{x}) \leq f_i(\overrightarrow{y}) \ , \ \varepsilon \in \Re^+ \qquad (2.14)$$

De modo a considerar alguns aspectos ignorados pela dominância de Pareto, como a quantidade e relevância das melhoras e pioras entre os objetivos, Benedetti, Farina e Gobbi [Benedetti, Farina e Gobbi 2006] apresentam a (1-k)-dominância. Inicialmente, definem funções que contam quantos objetivos de uma solução $\overrightarrow{x}$ são melhores (b_t), iguais (e_q) ou piores (w_s) que os objetivos de uma

solução $\overrightarrow{y}$, formalmente definidas pela equação (2.15).

$$\begin{aligned} b_t(\overrightarrow{x},\overrightarrow{y}) &\triangleq |\{i \in \aleph \,|\, i \leq M \wedge \mathfrak{f}_i(\overrightarrow{x}) < \mathfrak{f}_i(\overrightarrow{y})\}| \\ e_q(\overrightarrow{x},\overrightarrow{y}) &\triangleq |\{i \in \aleph \,|\, i \leq M \wedge \mathfrak{f}_i(\overrightarrow{x}) = \mathfrak{f}_i(\overrightarrow{y})\}| \\ w_s(\overrightarrow{x},\overrightarrow{y}) &\triangleq |\{i \in \aleph \,|\, i \leq M \wedge \mathfrak{f}_i(\overrightarrow{x}) > \mathfrak{f}_i(\overrightarrow{y})\}| \\ \text{onde: } M &= b_t + e_q + w_t \end{aligned} \tag{2.15}$$

Assim, diz-se que uma solução $\overrightarrow{x}$ (1-k)-domina outra solução $\overrightarrow{y}$, representada pela relação $\overrightarrow{x} \preceq_{1-k} \overrightarrow{y}$ se e apenas se a condição da equação (2.16) é satisfeita.

$$\overrightarrow{x} \preceq_{1-k} \overrightarrow{y} : \begin{cases} e_q & < M \\ b_t & \geq \lceil \frac{M-e_q}{k+1} \rceil \end{cases}, \; 0 \leq k \leq 1 \tag{2.16}$$

Neste caso, quando $k = 0$ a (1-k)-dominância equivale à dominância de Pareto. Uma definição difusa de otimalidade também foi apresentada, mas só é utilizada como um critério de seleção a posteriori [Benedetti, Farina e Gobbi 2006].

Kokunem e Lampinen [Kukkonen, Member e Lampinen 2007] definiram uma relação de preferência baseada no ordenamento de um conjunto de soluções de acordo com cada objetivo separadamente e uma função de agregação que calcula ordenamento final de cada solução. Essa relação é denominada ranking-dominância, e pode ser usada em problemas com muitos objetivos. Uma função de agregação $\mathfrak{agg}$ pode ser usada para gerar o ordenamento final *Ragg* de uma solução $\overrightarrow{x}$, conforme a equação (2.17). Uma das funções de agregação sugeridas é o somatório, de modo que o ordenamento final $Ragg_{sum}$ é dado pela equação (2.18), onde $\mathfrak{rnk}$ fornece o ordenamento da solução em relação a um objetivo específico.

$$Ragg(\overrightarrow{x}) = \mathfrak{agg}(\mathfrak{rnk}(f_1(\overrightarrow{x})), \mathfrak{rnk}(f_2(\overrightarrow{x})), ..., \mathfrak{rnk}(f_M(\overrightarrow{x}))) \tag{2.17}$$

$$Ragg_{sum}(\overrightarrow{x}) = \sum_{i=1}^{M} \mathfrak{rnk}(f_i(\overrightarrow{x})) \tag{2.18}$$

Diz-se então que uma solução $\overrightarrow{x}$ ranking-domina outra solução $\overrightarrow{y}$ se ela atende à condição da equação (2.19) e usa-se a notação $\overrightarrow{x} \prec_{rnk} \overrightarrow{y}$ para representá-la.

$$\overrightarrow{x} \prec_{rnk} \overrightarrow{y} \rightarrow Ragg(\overrightarrow{x}) < Ragg(\overrightarrow{y}) \tag{2.19}$$

Os autores mostram que $\overrightarrow{x} \prec_p \overrightarrow{y} \rightarrow \overrightarrow{x} \prec_{rnk} \overrightarrow{y}$ quando a função somatório é usada como função de agregação, mas o mesmo não é verdade para outras

funções, mesmo as sugeridas pelos autores. Também mostram que $\overrightarrow{x} \prec_{rnk} \overrightarrow{y} \nrightarrow \overrightarrow{x} \prec_p \overrightarrow{y}$, mesmo para a função de somatório, ou seja, nem sempre a seleção baseada na ranking-dominância avança em direção à fronteira de Pareto, e mais pesquisas são necessárias para melhorar essa relação de preferência [Kukkonen, Member e Lampinen 2007].

Le e Landa-Silva [Le e Landa-Silva 2007] propuseram a dominância de volume, chamada aqui de v-dominância, que compara duas soluções em relação ao volume que cada solução domina e também considera o volume que ambas soluções dominam simultaneamente [Le e Landa-Silva 2007], e parece claramente baseada no hipervolume. O volume dominado por uma solução $\overrightarrow{x}$ com base num ponto de referência $\overrightarrow{r} = [r_1, r_2, ..., r_M]$ é definido na equação (2.20). Entretanto, a dominância é definida com base no volume relativo que é dominado simultaneamente por duas soluções sendo comparadas, $\overrightarrow{x}$ e $\overrightarrow{y}$, formalmente definido na equação (2.21).

$$V_{\overrightarrow{x}} = \prod_{i=1}^{M} (f_i(\overrightarrow{x}) - r_i) \tag{2.20}$$

$$SV_{\overrightarrow{x},\overrightarrow{y}} = \prod_{i=1}^{M} (min(f_i(\overrightarrow{x}), f_i(\overrightarrow{y})) - r_i) \tag{2.21}$$

Então, pode-se dizer que uma solução $\overrightarrow{x}$ v-domina uma solução $\overrightarrow{y}$, representada pela relação $\overrightarrow{x} \preceq_v \overrightarrow{y}$ se:

$$\begin{aligned} \overrightarrow{x} \preceq_v \overrightarrow{y} : (V_{\overrightarrow{y}} = SV_{\overrightarrow{x},\overrightarrow{y}} \wedge V_{\overrightarrow{x}} > SV_{\overrightarrow{x},\overrightarrow{y}}) \vee \\ \vee (V_{\overrightarrow{x}} > V_{\overrightarrow{y}} > SV_{\overrightarrow{x},\overrightarrow{y}} \wedge r_{\overrightarrow{x},\overrightarrow{y}} > rSV) \\ Onde : r_{\overrightarrow{x},\overrightarrow{y}} = \frac{V_{\overrightarrow{x}} - V_{\overrightarrow{y}}}{SV_{\overrightarrow{x},\overrightarrow{y}}} \end{aligned} \tag{2.22}$$

Zou et alli [Zou et al. 2008] apresentam a l-dominância para tratar da otimização de muitos objetivos. Diz-se que uma solução $\overrightarrow{x}$ l-domina outra solução $\overrightarrow{y}$ se e apenas se ela atende a condição da equação (2.23), e usa-se a notação $\overrightarrow{x} \preceq_l \overrightarrow{y}$ para representá-la (b_t e w_s correspondem à quantidade de soluções que são melhores ou piores que $\overrightarrow{y}$, como definido na equação (2.15)). Um aspecto importante da l-dominância é que ela é proposta como solução quando muitos objetivos são considerados. Entretanto, os autores assumem que todos os objetivos são igualmente importante e que, quando há muitos objetivos e alguns

não são igualmente importantes devem ser simplesmente eliminados da otimização.

$$\overrightarrow{x} \preceq_l \overrightarrow{y} : \begin{cases} b_t - w_s = L > 0 \\ \| f^{'}(\overrightarrow{x}) \|_p \leq \| f^{'}(\overrightarrow{y}) \|_p \end{cases} \tag{2.23}$$

Embora boa parte das pesquisas utilize a dominância de Pareto ($\prec_p$) como critério de escolha das soluções preferíveis, diversas alternativas têm sido propostas para contornar algumas de suas limitações, principalmente em relação à otimização de muitos objetivos e à preferência do projetista. Contudo, uma análise comparativa dessas alternativas que utilize um critério de avaliação consistente e representativo, ainda não foi apresentada. Nesta tese ainda foi utilizada a dominância de Pareto nos experimentos realizados. Contudo, a forma de dominância foi modelada de modo a permitir sua alteração de forma transparente ao resto do sistema. Com isso, novas formas de dominância podem ser facilmente testadas.

2.2.3 Preferências do Projetista

O conjunto de soluções preferíveis, ou não-dominadas, pode ser infinito. O conjunto de soluções encontrado pelo otimizadores é, portanto, apenas um sub-conjunto do conjunto de soluções preferíveis. Entretanto, esse sub-conjunto pode ser, e até deve ser, composto por muitas soluções, de modo a melhor representar a fronteira de Pareto. Todavia, do ponto de vista do projetista, obter um conjunto grande de possíveis soluções pode não ser útil para a tomada de decisão, e escolher entre elas a melhor solução pode consumir tempo e realmente não ser fácil [Ishibuchi et al. 2009].

Recentemente tem havido crescente interesse na busca de soluções considerando as preferências do projetista [Said, Bechikh e Ghédira 2010], que visam tanto escolher uma solução final quanto obter soluções que não sejam uniformemente distribuídas sobre toda a fronteira de Pareto, mas sim direcionadas para uma parte da fronteira que seja preferida pelo projetista, conhecida como região de interesse (ROI - *Region of Interest*) [Deb 2001, Purshouse e Fleming 2007, Said, Bechikh e Ghédira 2010][1]. Conforme a classi-

[1] no contexto de sistemas embarcados, o projetista poderia especificar que ele está interessado em soluções cuja área física seja aproximadamente $3cm^2$ e que a frequência de operação seja cerca de 40MHz, por exemplo.

ficação definida por Miettinen [Miettinen 1999], em relação à incorporação de preferências do projetista no processo de otimização, abordagens podem ser divididos em: (1) Sem preferência, em que as preferências não são consideradas; (2) *A priori*, em que as preferências são definidas antes do início do processo de otimização; (3) *A posteriori*, em que as preferências são usadas depois da otimização ter gerado o conjunto de soluções preferíveis; e (4) Interativos, em que as preferências são fornecidas interativamente, durante a otimização.

A maioria dos MOEA (MOEA - *Multi Objective Evolutionary Algorithm*) pertencem à categoria *a posteriori* [Said, Bechikh e Ghédira 2010]. Dentre as abordagens *a posteriori*, incluem-se [Parreiras, Maciel e Vasconcelos 2006, Ishibuchi et al. 2009, Parreiras e Vasconcelos 2009]. No trabalho de Ishibushi et alli [Ishibuchi et al. 2009], por exemplo, é proposta uma estratégia para minimizar a quantidade de soluções apresentadas e, simultaneamente, maximizar o indicador de hipervolume. Esses são os dois objetivos do otimizador utilizado. Destas, apenas um pequeno conjunto de soluções escolhidas são apresentadas ao projetista, que escolhe uma. Então, numa segunda etapa, soluções mais próximas[2] à escolhida são apresentadas, e ele pode analisar essas novas soluções e atualizar sua escolha. Essa abordagem é interessante por incluir a preferência no conjunto de objetivos do otimizador; entretanto, peca pela escolha sucessiva das melhores soluções.

Dentre as abordagens *interativas*, pode-se citar [Ishibuchi, Tsukamoto e Nojima 2007]. Embora as estratégias interativas tenham aplicabilidade, elas não são uma opção viável à exploração do espaço de projeto em sistemas embarcados, pois o processo de otimização (exploração do espaço de projeto) pode ser muito lento (várias horas ou dias) e não pode-se esperar que o projetista esteja presente durante esse tempo para fazer escolhas interativas. A abordagem mais adequada nesse cenário é a de preferências *a priori*.

As abordagens *a priori* incluem [Zitzler, Brockhoff e Thiele 2007, Wang e Tai 2007, Bader e Zitzler 2008, Auger et al. 2009, Rachmawati e Srinivasan 2009] e normalmente incluem a alteração da formulação dos objetivos de forma a guiar a otimização a uma região predefinida e codificada na formulação dos objetivos. Nesta tese proponho a utilização de uma abordagem *a priori* integrada a um indicador de avaliação de otimizadores,

[2] a proximidade pode ser avaliada através da distância euclidiana, do indicador de hipervolume ou outras técnicas.

o *weighted hypervolume* (hipervolume ponderado), definido inicialmente por Bader e Zitzler et alli [Bader e Zitzler 2008], por permitir que o projetista informe suas preferências na forma de distribuições de probabilidade, por essas preferências alterarem a própria forma de seleção das soluções preferíveis e por permitir operar sobre problemas de otimização com muitos objetivos (tipicamente, algumas dezenas). A próxima seção, e mais especificamente a subseção 2.3.2 apresenta tal indicador. A figura 2.4 apresenta alguns exemplos de como essas distribuições de probabilidade afetam a distribuição das soluções encontradas. Nessa figura, os eixos correspondem a dois objetivos sendo minimizados, a linha contínua corresponde à fronteira de Pareto e as soluções não-dominadas encontradas pelo otimizador correspondem aos círculos sobre a fronteira e que, sem preferências do projetista, estariam distribuídas de maneira mais ou menos uniforme sobre a fronteira. As distribuições de probabilidade são representadas por linhas de contorno em intervalos de 10% do peso máximo. A figura 2.4(a) mostra o efeito da mudança do único parâmetro da distribuição exponencial, que foi a distribuição usada nesse caso, enquanto a figura 2.4(b) mostra o efeito da variação do desvio-padrão da distribuição normal, usada nesse caso e também nas figuras 2.4(c) e 2.4(d), que mostram os efeitos da mudança da direção e da média dessa distribuição, respectivamente. Em todos os casos, os efeitos correspondem à concentração das soluções encontradas sobre a ROI do projetista, especificada através da distribuição de probabilidade escolhida. Um dos problemas apresentados pelo autor é que o posicionamento da distribuição de probabilidade que representa os pesos pode depender de algum conhecimento prévio sobre a fronteira de Pareto, pois se ela for especificada muito longe da região viável, ela basicamente não terá efeito [Auger et al. 2009], como mostra a figura 2.4(d).

2.3 Avaliação de Otimizadores Multiobjetivo

O processo de avaliar diferentes algoritmos de otimização multiobjetivo segue basicamente algumas etapas comuns [Huband et al. 2006]: (1) Escolher os algoritmos a comparar; (2) Escolher um conjunto de problemas de teste; (3) Escolher um conjunto de indicadores de qualidade para comparar os resultados produzidos pelos algoritmos; (4) Obter resultados para cada algoritmo em cada problema de teste; (5) Gerar os indicadores a partir dos resultados e comparar os dados, para finalmente, (6) Tirar conclusões. Nesta seção, diferentes conjuntos

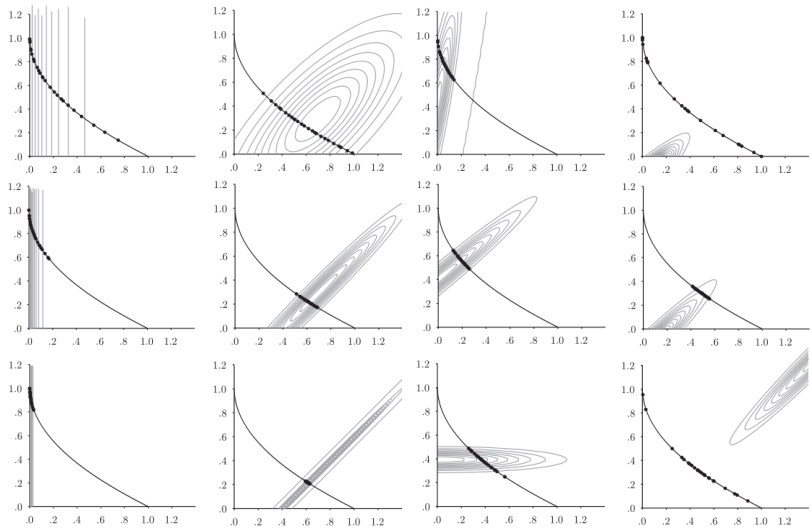


Figura 2.4: *Inclusão de preferências do projetista no indicador de hipervolume ponderado. As figuras apresentam o efeito da mudança de parâmetros das distribuições de probabilidade usadas na distribuição das soluções encontradas, que concentram-se sobre as áreas que representam as preferências do projetista.*
Fonte: [Auger et al. 2009]

de testes e indicadores de qualidade são apresentados e analisados.

2.3.1 Conjuntos de Problemas de Teste

A comparação entre otimizadores exige uso de conjuntos de teste comuns como *benchmarks*. Até alguns anos, muitos conjuntos de teste não tinham sido rigorosamente analisados [Huband et al. 2006], o que tornava difícil tirar conclusões sobre as vantagens e limitações de cada algoritmo. É necessário que conjuntos de testes sejam projetados para avaliar condições específicas dos algoritmos e possuam características adequadas para esse propósito. Deb et alli [Deb et al. 2002] apresentam cinco características gerais que devem estar presentes num conjunto de problemas de teste de otimizadores multiobjetivo: (1) Ser

fácil de construir; (2) Ser escalável em relação à quantidade de variáveis de decisão; (3) Ser escalável em relação à quantidade de funções-objetivo; (4) Ter a localização e formato da fronteira de Pareto bem conhecidos; e (5) Introduzir algum obstáculo aos otimizadores convergirem à fronteira de Pareto e/ou terem soluções bem distribuídas. Num trabalho posterior [Deb et al. 2006], eles incluem ainda outra característica: (6) Apresentar os obstáculos comuns a problemas reais.

As características (1) e (4) podem ser obtidas a partir de diferentes abordagens para construção de problemas de teste, como as sugeridas por Deb et alli [Deb et al. 2002]. As características (2) e (3) podem ser obtidas através do mapeamento de variáveis e funções, respectivamente[3]. As características (5) e (6) dependem, basicamente, da forma (*landscape*) do conjunto de Pareto e do contexto do problema. Entretanto, conjuntos de testes "artificiais" possuem vantagens sobre problemas reais no que se refere à aplicação dos algoritmos testados [Huband et al. 2006]. Desse modo, os obstáculos que são introduzidos aos otimizadores refletem-se basicamente na forma do conjunto e da fronteira de Pareto.

Diferentes formas podem ser utilizadas para introduzir diferentes dificuldades aos otimizadores. Fronteiras convexas (figura 2.5(a)) são as que apresentam maior facilidade para encontrar soluções não-dominadas, enquanto fronteiras não-convexas dificultam a convergência em várias técnicas clássicas (figura 2.5(b)). Nas fronteiras em que a distribuição das soluções não é uniforme, mas está concentrada sobre uma pequena região da fronteira (figura 2.5(c)), os otimizadores têm maior dificuldade em encontrar soluções sobre toda a fronteira. Fronteiras desconexas exigem que os otimizadores mantenham parte da população em cada segmento da fronteira (figura 2.5(d)). O conjunto de Pareto pode apresentar várias fronteiras com mínimos locais, o que pode fazer o otimizador ficar preso numa fronteira local (figura 2.5(e)). Essas formas básicas que introduzem dificuldades podem ser ajustadas (através da manipulação das equações das funções-objetivo) para acentuar ou atenuar suas características, e podem ainda ser combinadas entre si para produzir problemas mais difíceis de resolver. Na figura 2.5(f) a fronteira de Pareto é desconexa e apresenta vários mínimos locais. Na figura 2.5(g), o conjunto de Pareto é conexo, mas sua fronteira é desconexa. Essa forma apresenta uma dificuldade maior para os otimizadores do que a forma da figura 2.5(d), devido à manutenção de populações nos diferentes segmentos

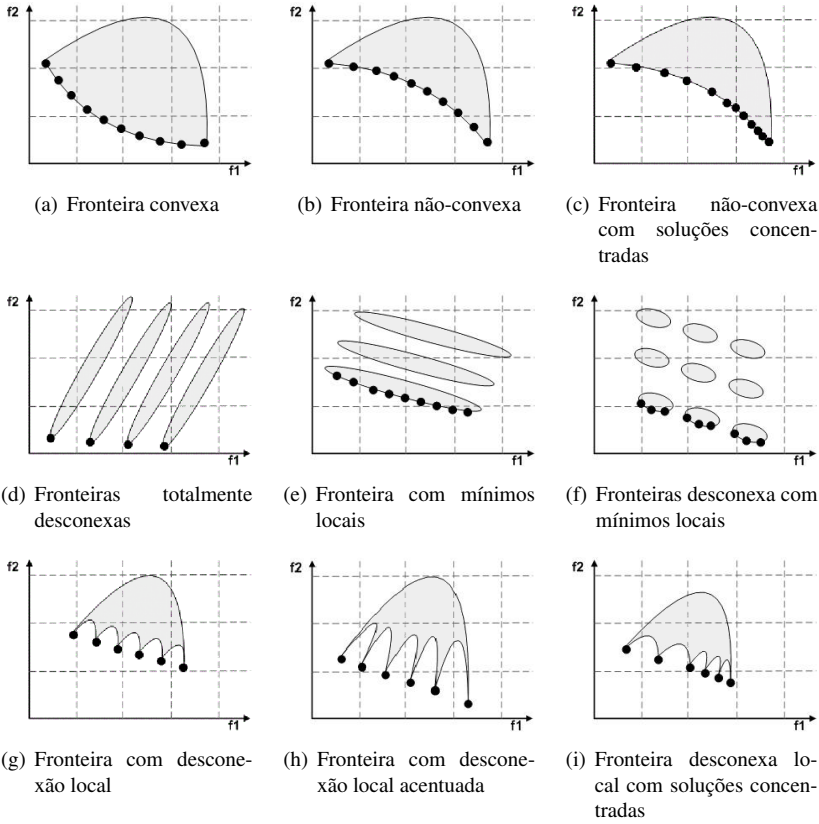
[3]como exemplo, considere uma função-objetivo $f_i(\vec{x})$, $x = [x_1, x_2, ..., x_m]$. Podemos transformar essa função de m variáveis numa função de $m+n$ variáveis através de uma função de transformação $x_1 = g(y_1, y_2, ..., y_n)$.

descontínuos após uma área contínua. Como citado, o problema inserido por essa forma pode ser acentuado ou atenuado, e a figura 2.5(h) apresenta a mesma forma, mas que apresenta maior dificuldade aos otimizadores para encontrar a fronteira de Pareto, apenas pela manipulação de um parâmetro. A figura 2.5(i) combina duas dificuldades: fronteira desconexa com soluções concentradas. Essas formas podem representar grandes desafios a qualquer otimizador.

Diferentes funções-objetivo podem ser usadas para gerar essas formas. Huband et alli [Huband et al. 2006] apresentam algumas características de funções-objetivo que também introduzem obstáculos à otimização. Elas incluem funções com *regiões planas*, que são aquelas em que variações nos parâmetros não produzem mudanças significativas nas funções-objetivo, de modo que os otimizadores não tem informação de gradiente nessas regiões e não "sabem" para onde convergir. Diz-se que é *ótimo isolada* a função-objetivo no qual boa parte de seu conjunto de Pareto é plano, não fornecendo informação útil sobre o ponto ótimo. A figura 2.6(a) apresenta um exemplo dessa forma. Dificuldades também são introduzidas com a *multimodal*, que é a função-objetivo que apresenta múltiplos mínimos locais. Funções *multimodais* podem fazer com que os otimizadores fiquem presos a mínimos locais, e a figura 2.6(b) apresenta um exemplo desta forma. Em contrapartida, *unimodal* é uma função-objetivo que apresenta um único mínimo local. Uma função *deceptiva* é a função-objetivo multimodal no qual a maior parte do espaço de busca leva ao mínimo local deceptivo. Esse tipo de função também dificulta os otimizadores encontrarem o mínimo global. A figura 2.6(c) apresenta um exemplo desta forma.

Surge então a questão de como criar conjuntos de problemas de testes que atendam essas características. Deb et alli [Deb et al. 2002] sugerem três abordagens principais pelas quais funções de teste podem ser geradas:

- Múltiplas funções de um único objetivo: é fácil de construir, mas o conjunto de Pareto resultante pode ser difícil de achar, além de ser falha nas características (2) a (6).
- Bottom-up: a forma exata do conjunto de pareto pode ser definida pelo projetista a priori e a escalabilidade é controlada, mas o método não é tão simples de aplicar e ainda cabe ao projetista inventar funções que atendam às características (5) e (6); ou
- Superfície com restrições: a construção é mais simples que o método *bottom-up*, mas o conjunto de Pareto não é tão fácil de expressar matemati-



(a) Fronteira convexa

(b) Fronteira não-convexa

(c) Fronteira não-convexa com soluções concentradas

(d) Fronteiras totalmente desconexas

(e) Fronteira com mínimos locais

(f) Fronteiras desconexa com mínimos locais

(g) Fronteira com desconexão local

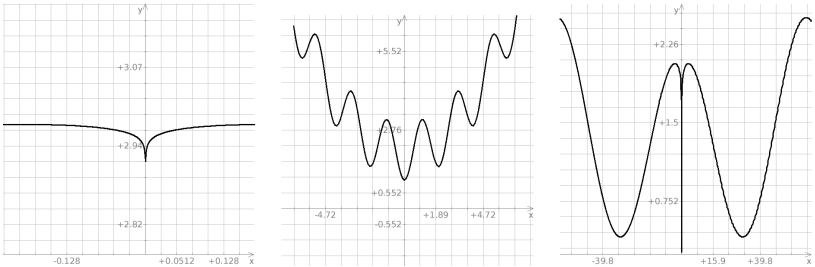
(h) Fronteira com desconexão local acentuada

(i) Fronteira desconexa local com soluções concentradas

Figura 2.5: *Algumas formas da fronteira do conjunto de Pareto. Diferentes formas introduzem dificuldades diferentes para os otimizadores, e que podem ser acentuadas ou atenuadas por parâmetros nas funções-objetivo.*
Fonte: Adaptado de [Deb 2009]

camente nem de visualizar, exige dos algoritmos de otimização estratégias para tratar restrições, e apresenta problemas com as características (4), (5) e (6).

Nesta tese é usado o conjunto de problemas de teste ZDT (Zitzler-Deb-



(a) Plana com ótimo isolado (b) Multimodal (c) Deceptive

Figura 2.6: *Algumas formas de funções-objetivo. Exemplos de formas que introduzem dificuldades para os otimizadores*

Thiele) para avaliação do modelo de otimização. A especificação completa das funções de teste desse conjuntos é apresenta na seção 7.2.

2.3.2 Indicadores de Qualidade

Indicadores de qualidade são utilizados para avaliar o desempenho de um otimizador ou para comparar o desempenho relativo de diferentes otimizadores, mapeando um conjunto de soluções a um número real. Apesar dos avanços recentes, a comparação de algoritmos de otimização multiobjetivo ainda é uma questão a ser considerada [Bradstreet, Barone e While 2009]. A escolha do indicador depende da afirmação que se deseja fazer. Normalmente, deseja-se afirmar que o conjunto de soluções $\mathscr{A}$ domina o conjunto $\mathscr{B}$. Importante também é saber o quanto melhor ou pior um conjunto é em relação a outro. Mesmo para conjuntos incomparáveis ($\mathscr{A} \parallel \mathscr{B}$), seria interessante saber em quais aspectos um conjunto é melhor que outro. Indicadores podem avaliar a proximidade das soluções em relação à fronteira de Pareto, a diversidade e distribuição das soluções, ou então avaliar esses dois aspectos simultaneamente [Deb et al. 2002, Beume et al. 2009]. Dado que boas soluções são caracterizadas por ambos aspectos, é desejável utilizar um único indicador que os avalie simultaneamente.

Muitos indicadores foram propostos, e discussões deles podem ser encontradas em [Zitzler et al. 1998, Knowles e Corne 2002, Zitzler et al. 2003]. Sem-

pre que desenvolve-se um indicador I qualquer, estabelece-se uma relação de preferência de soluções, como definido na seção 2.2, de modo que $\mathscr{A} \prec_I \mathscr{B} := I(\mathscr{A}) > I(\mathscr{B})$. Ressalta-se apenas que a escolha do indicador adequado é muito importante, e que alguns indicadores encontrados na literatura foram posteriormente classificados como não completos e não compatíveis. Um indicador de qualidade I é dito *compatível* em relação a uma relação de preferência $\prec$ se, e apenas se, $\forall \mathscr{A}, \mathscr{B} \mid \mathscr{A} \prec \mathscr{B} \rightarrow I(\mathscr{A}) > I(\mathscr{B})$ [Knowles e Corne 2002]. Em geral, busca-se um indicador que seja Pareto-compatível, ou seja, $\forall \mathscr{A}, \mathscr{B} \mid \mathscr{A} \prec_P \mathscr{B} \rightarrow I(\mathscr{A}) > I(\mathscr{B})$

Um dos indicadores que tem recebido grande destaque nos últimos anos é o hipervolume. O hipervolume foi introduzido por Zitzler e Thiele [Zitzler e Thiele 1999], e foi inicialmente chamado de "tamanho do espaço coberto" (*size of the covered space*), e apresenta algumas características interessantes, como integrar, num único valor numérico, informações sobre a proximidade da fronteira de Pareto e a distribuição das soluções [Auger et al. 2009], ser Pareto-completo e Pareto-compatível. O hipervolume representa a porção do espaço M-dimensional que é dominado por um conjunto de soluções. Assim, um conjunto de soluções com maior hipervolume representa um conjunto melhor que outro com menor hipervolume [Bradstreet, Barone e While 2009].

Dado um conjunto $\mathscr{Q}$ de pontos no espaço positivo M-dimensional $\Re^M_{\geq 0}$, o hipervolume corresponde ao espaço que é dominado por pelo menos um ponto no conjunto $\mathscr{Q}$, em relação a um ponto de referência $r = (r_1, r_2, ..., r_M) \in \Re^M$, conforme definido na equação (2.24). A figura 2.7 representa o hipervolume num espaço bidimensional $\Re^2$, ou seja, com dois objetivos. Nesse caso, ele corresponde à área sobre as soluções não-dominadas, em relação ao ponto de referência r. A figura 2.8 representa o hipervolume num espaço tridimensional $\Re^3$, em que ele corresponde ao volume dominado pelas soluções indicadas com números 1 a 6. Para espaços M-dimensionais o termo hipervolume é adequado.

$$\Pi^M = \left\{ x \in \Re^M_{\geq 0} : x \preceq y, \exists y \in \mathscr{Q} \right\} \tag{2.24}$$

O cálculo do hipervolume depende em grande parte da escolha do ponto de referência, e não pode estar nem muito distante nem muito próximo do conjunto de soluções, pois afeta a contribuição das soluções extremas [Ishibuchi et al. 2009]. Além disso, algumas implementações do hipervolume (ex. [Bader, Brockhoff e Zitzler 2011]) exigem a determinação também do ponto

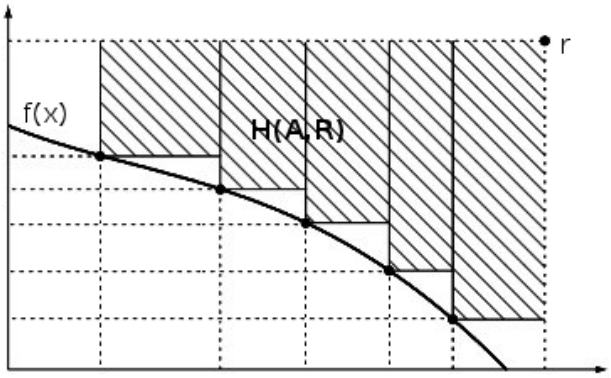


Figura 2.7: *Representação do hipervolume em 2D. O hipervolume H(A,R) corresponde à área que é dominada por um conjunto de soluções A, representado na figura por uma função f(x), delimitada por um ponto de referência R = r.*
Fonte: [Auger et al. 2009]

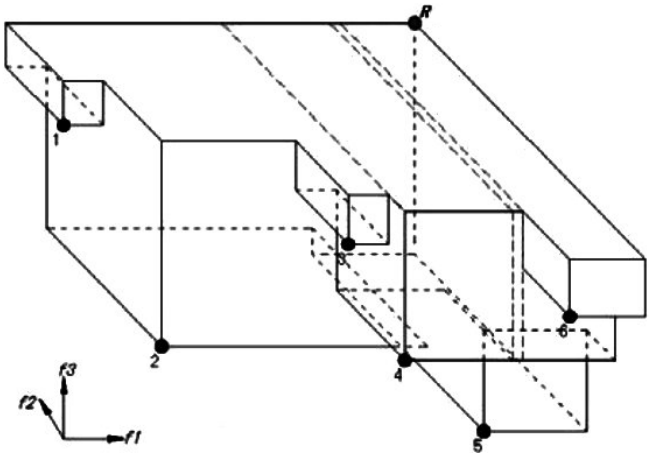


Figura 2.8: *Representação do hipervolume em 3D. O hipervolume corresponde ao volume dominado por um conjunto de soluções (pontos 1 a 6), delimitado por um ponto de referência R.*
Fonte: [Li et al. 2009]

de nadir e do ponto ideal[4] [Deb 2009].

Algumas pesquisas indicam que o hipervolume pode ser calculado com pelo menos $\Omega(n.log\,n)$ e no máximo $\mathcal{O}(n.log\,n + n^{M/2}.log\,n)$ comparações num conjunto de tamanho n em M dimensões [Beume et al. 2009]. Entretanto, é possível diminuir drasticamente a quantidade de comparações em cálculos consecutivos do hipervolume, ao considerar o cenário em que elas são realizadas. Como o hipervolume é calculado pelos otimizadores também para seleção ou elitismo, em geral um novo cálculo é necessário sempre que um ponto (uma solução) é incluído ou removido. Nesses casos, técnicas como a desenvolvida por Bradstreet [Bradstreet, Barone e While 2009] permitem a atualização do hipervolume sem seu recálculo completo, o que, conforme os autores, pode reduzir de 72% a 99% o tempo de cálculo do hipervolume (comparado ao tempo de recálculo completo no DTLZ). Basicamente, essas técnicas consistem em calcular a contribuição s_p de cada ponto p de um conjunto de soluções $\mathscr{S}$ para o hipervolume (H), definida como $s_p = H(\mathscr{S} \cup \{p\}) - H(\mathscr{S})$. Outras pesquisas [Bader e Zitzler 2008, Li et al. 2009] também utilizam o cálculo da contribuição individual de cada solução ao hipervolume para eliminar soluções no conjunto elitista (que é finito) ou para eliminar soluções com menor desempenho (no processo de seleção ambiental). Essas técnicas são muito importantes principalmente para a otimização de muitos-objetivos, pois o tempo de recálculo completo do hipervolume cresce exponencialmente com a quantidade de objetivos [Zou et al. 2008].

Um aperfeiçoamento do indicador de hipervolume foi proposto por Bader [Bader 2009] em 2009. Em sua abordagem, dentre alguns aperfeiçoamentos, como uma estratégia de amostragem diferenciada para cálculo do hipervolume, destaca-se a aplicação de distribuições sobre o hipervolume, formando um *bias* que altera os pesos das soluções não-dominadas encontradas (por isso o termo hipervolume ponderado). Com isso, é possível definir, *a priori*, usando essas distribuições, quais serão as melhores soluções (aquelas com maior peso). Com isso, esse indicador passa a ser também uma forma do projetista especificar suas preferências, como foi apresentado na seção 2.2.3. Além disso, o indicador de hipervolume pode ser usado também como estratégia de seleção das melhores soluções, transformando problemas de otimização multiobjetivo em otimização de

[4] O *ponto de nadir* representa o limite superior de cada função-objetivo no conjunto completo de Pareto. O *ponto ideal* corresponde ao valor mínimo de cada função-objetivo individualmente num espaço de M dimensões, e normalmente corresponde a uma solução não existente, pois as funções-objetivos são normalmente conflitantes.

um único objetivo: a maximização do próprio indicador de hipervolume, como será descrito na seção 3.3.2. Por suas características, sua versatilidade e possibilidade de uso como indicador de qualidade, estratégia de seleção, e como técnica de inclusão de preferências do projetista, o hipervolume é utilizado nesta tese.

2.4 Estratégias de Otimização Multiobjetivo

Diversas estratégias podem ser utilizadas para tratar um problema de otimização multiobjetivo, dentre as quais destacam-se as técnicas estocásticas. Como as funções-objetivo geralmente são complexas ou mesmo suas expressões matemáticas são desconhecidas, abordagens clássicas de otimização falham em determinar as soluções ótimas, ou levam muito tempo [Bader 2009]. Técnicas estocásticas facilitam a otimização de sistemas que são difíceis ou mesmo impossíveis de serem otimizados através de técnicas clássicas, quer seja porque há pouco conhecimento sobre o sistema em si, quer seja porque a quantidade de soluções possíveis torna a otimização exata impraticável. Além disso, MOP normalmente têm muitas soluções preferíveis, e não apenas uma solução ótima. Diversas técnicas estocásticas para otimização foram desenvolvidas, dentre as quais as mais comuns são:

- *Tabu Search* (Busca Tabu): É uma busca determinística pela vizinhança (baseada em gradiente) evitando mínimos locais. Possui uma memória adaptativa de curta duração para evitar ficar preso numa região já explorada.
- *Simulated Annealing* (Recozimento Simulado): Tem inspiração física, baseada na acomodação de menor nível de energia de partículas após seu aquecimento e resfriamento. Usa uma função de probabilidade (análoga à temperatura) que permite o movimento para uma solução pior com probabilidade decrescente (diminuição da temperatura) à medida que o sistema progride.
- *Ant Colony* (Colônia de Formigas): De inspiração biológica, visa imitar o comportamento de formigas reais, que são capazes de encontrar caminhos mínimos usando trilhas de feromônios.
- *Evolutionary Algorithm* (Algoritmos Evolucionários): São uma definição estendida dos algoritmos genéticos, inspirados na biologia evolucionária.

Representam soluções como indivíduos que são selecionados para reprodução conforme sua aptidão ao ambiente.

- *Particle Swarm* (Enxame de Partículas): Tem demonstrado ser muito eficiente em vários tipos de problemas. Representam soluções como partículas aleatórias que se movem no espaço de projeto em busca dos mínimos globais. A cada partícula é atribuído um vetor de direção e uma velocidade.

Das técnicas estocásticas usadas para otimização, tem recebido bastante destaque os algoritmos evolucionários e neles, os algoritmos genéticos. A primeira pesquisa com algoritmos genéticos para MOP data de meados da década de 1980 [Zitzler 1999, Coello 2009], com os trabalhos de David Schaffer e a criação de um mecanismo de seleção moficado para MOP, o (VEGA - *Vector Evaluated Genetic Algorithm*) [Coello 2006]. Dentre algumas características favoráveis apresentadas pelos algoritmos evolucionários, destacam-se as seguintes: (1) É conceitualmente muito simples; (2) Pode ser aplicada a praticamente qualquer problema de otimização multiobjetivo; (3) Não possui muitos pressupostos e restrições matemáticas; (4) Pode representar soluções inviáveis, o que facilita encontrar soluções na fronteira viável/inviável; (5) Pode ser combinada com outras técnicas heurísticas ou exatas; (6) É altamente paralelizável e permite encontrar muitas soluções numa única iteração, já que trabalha com uma população; (7) Pode ser utilizada para otimizar a si mesma, adaptando seus parâmetros para alcançar a solução ótima mais rapidamente.

Por essas características, esta tese foca no uso de técnicas de algoritmos evolucionários para otimização multiobjetivo e exploração do espaço de projeto de sistemas embarcados.

CAPÍTULO 3

ALGORITMOS EVOLUCIONÁRIOS PARA OTIMIZAÇÃO MULTIOBJETIVO

Não é o mais forte que sobrevive, nem o mais inteligente, mas o que melhor se adapta às mudanças.
—CHARLES DARWIN

A seleção natural é um relojoeiro cego; cego porque não antecipa, não planeja consequências, não tem um fim em vista, embora os seus resultados nos impressionem vivamente e pareçam ser a de um mestre relojoeiro, envolvendo design e planejamento.
—RICHARD DAWKINS (The Blind Watchmaker, 1986)

Não há qualquer força espiritual guiando-nos, palpitando, pesando, pululando, protoplásmica, qual geleia mística. A vida é apenas bytes, bytes, e bytes, ou informação digital.
—RICHARD DAWKINS (River out of Eden, 1995)

Este capítulo apresenta questões associadas aos algoritmos evolucionários usados para otimização multiobjetivo, e é dividido em 3 seções. A seção 3.1 apresenta o processo bioquímico associado à genética, e que fornece suporte à evolução das espécies, e visa descrever alguns elementos que não são contemplados adequadamente nos algoritmos evolucionários. A seção 3.2 apresenta uma breve fundamentação e classificação dos algoritmos evolucionários. A seção 3.3 foca em questões fundamentais para os problemas de otimização multiobjetivo, e apresenta as principais heurísticas para codificação, seleção, aptidão, diversidade, e variação.

3.1 Inspiração Biológica

Algoritmos evolucionários são uma técnica de otimização com inspiração na teoria da evolução por seleção natural, de Charles Darwin. A ideia central

dessa teoria é a seleção natural: os mais adaptados às condições do meio ambiente sobrevivem e se reproduzem e a cada geração os que se reproduzem são, preferencialmente, aqueles que possuem melhores condições de adaptação ao meio ambiente. Entretanto, seu autor não conhecia o processo bioquímico através do qual a evolução ocorre, o que só começou a ser conhecido na metade do século XX. Ainda hoje o entendimento do processo bioquímico não é completo, e novas descobertas têm sido apresentadas com frequência nas últimas décadas. Como tal processo é a base sobre a qual a evolução opera e os algoritmos evolucionários são, em última instância, representações computacionais abstratas desse processo, parece ser importante o seu entendimento.

A vida em nosso planeta é muito diversificada, de forma que é difícil apresentar uma descrição única que represente bem o processo bioquímico da genética, de bactérias a seres humanos, de procariontes e eucariontes[1]. Contudo, de forma geral e bem superficial, um indivíduo pode ser composto por um ou mais cromossomos, que constituem-se de fitas contínuas de DNA associadas a moléculas de histonas, que são proteínas básicas [Lewin 2007]. O DNA (DNA - *Deoxyribonucleic Acid*) é uma macromolécula contendo milhões de nucleotídeos, as menores unidades de informação genética. Boa parte desses nucleotídeos não traz qualquer contribuição ao fenótipo e não tem qualquer efeito prático para o indivíduo (embora possa tê-lo para muitas gerações de indivíduos). Entretanto, algumas sequências de nucleotídeos formam genes, que normalmente tem sequências codificantes desconexas chamadas exons e que são separadas por regiões sem efeito, denominadas íntrons [Lodish et al. 2000]. A unidade básica de informação na região codificante dos genes é o códon[2], composto por 3 nucleotídeos em sequência, e que serão traduzidos num único aminoácido numa proteína, conforme um código robusto e altamente redundante [Novozhilov, Wolf e Koonin 2007]. O padrão periódico do DNA em exons é um fenômeno bem conhecido, e é determinado pelas frequências de uso dos códons [Eskesen et al. 2004]. Os genes são precedidos por regiões promotoras[3] que identificam o início do gene e que ajudam a definir as condições nas quais o gene será expresso (transcrito em RNA). Assim, embora um indivíduo tenha muitos genes, poucos são transcri-

[1] procariontes são organismos sem membrana nuclear, em que o DNA fica no citoplasma. Eucariontes possuem tal membrana, isolando o DNA no núcleo celular.

[2] um *códon* é uma sequência composta por 3 nucleotídeos e que forma o código que corresponde a um único aminoácido na tradução de uma proteína.

[3] regiões promotoras incluem o ORF (ORF- Open Read Frame), que define a posição de início de transcrição do gene, mas incluem ainda outros elementos.

tos num dado instante e nem todos os genes transcritos são traduzidos em proteína; alguns deles são usados apenas para regulação da expressão genética [Hermsen, Ursem e Wolde 2010]. Assim, a informação genética modifica o ambiente bioquímico que, por sua vez, atua na regulação da expressão de outros genes.

Além disso, podem ocorrer erros na transcrição dos genes, bem como na replicação do DNA (as mutações); entretanto, as taxas de mutação não são constantes, e variam entre genes e mesmo entre códons num único gene, devido à presença de diversos elementos epigenéticos[4] e também devido à seleção natural. Em geral, os genes mais importantes e sob maior pressão da seleção natural são menos suscetíveis a mutações [Lewin 2007]. Mesmo num único gene, os códons que codificam aminoácidos que fazem parte dos sítios ativos[5] de uma proteína tendem a ser muito menos mutáveis, para que sua funcionalidade não seja perdida.

Embora nos organismos mais simples (procariontes) haja apenas uma sequência contínua de DNA (circular), nos organismos mais complexos o DNA está disposto em várias sequências independentes, os cromossomos, e genes cuja expressão está correlacionada costumam estar no mesmo cromossomo. Nos indivíduos diploides há, na realidade, um par de cada cromossomo, de forma que o indivíduo pode possuir duas versões diferentes de cada gene numa dada posição de um cromossomo (locus), que são os alelos. Genes alelos competem entre si para serem transcritos, assim como indivíduos competem para se reproduzirem, sendo que alguns genes são mais ou menos aptos que outros, dependendo do ambiente bioquímico no qual eles se encontram. Há teorias que afirmam que é o gene, e não o indivíduo, a entidade básica de seleção natural [Dawkins 1976].

Após algumas transformações, o RNA é traduzido em aminoácidos que formam as proteínas, outro tipo de macromolécula. Depois de formadas, as proteínas sofrem muitas transformações, formando estruturas de mais alto nível (secundária a quaternária), mudando sua função, interagindo com muitas outras proteínas e gerando novas proteínas e diferentes compostos químicos, definidos por redes (vias) metabólicas[6] [Lodish et al. 2000]. Por fim, pode-se dizer que o conjunto das diversas redes metabólicas é que define efetivamente o fenótipo do in-

[4] a *epigenética* refere-se a mudanças moleculares que regulam a função de genes, mas que não envolvem mudanças no material genético.

[5] um *sítio ativo* é uma região de uma proteína que participa da interação dessa proteína com outros compostos, e é responsável pela manutenção da funcionalidade da proteína no organismo.

[6] uma *rede metabólica* é uma série de reações químicas onde uma reação fornece o substrato da reação seguinte sendo a reação seguinte dependente da anterior.

divíduo, mas não parece que seja unicamente o fenótipo que determina a aptidão do indivíduo à sobrevivência e à reprodução, devido a muitos outros fatores ambientais e mesmo culturais. Portanto, há muitos níveis adicionais de tradução do genótipo até o fenótipo e sua aptidão para reprodução. Embora esse possa parecer um fluxo unidirecional, ele é, na realidade, iterativo, uma vez que as proteínas resultantes das redes metabólicas, bem como também algumas moléculas de RNA, agem também como reguladoras da expressão genética, ou seja, ditam quais genes serão transcritos [Lewin 2007]. Também há casos em que o RNA pode ser transformado de volta em DNA (cDNA) e incluído no material genético[7].

Os modelos evolucionários atuais não parecem refletir esse cenário biológico real. De modo geral (mas com exceções que merecem destaque), os indivíduos ainda são representados como há cerca de 40 anos, ou seja, como um único cromossomo composto unicamente por genes que contêm uma única informação e que são sempre transcritos para formar diretamente o fenótipo dos indivíduos. Os operadores genéticos também pouco mudaram, e a maioria dos modelos evolucionários ainda utiliza apenas a mutação simples e a recombinação de n blocos (normalmente, com $n=2$), nos quais o projetista ainda define empiricamente suas taxas de aplicação, que são iguais para todos os genes de todos os indivíduos de todas as gerações. Ainda de modo geral, pode-se dizer que os modelos abstraem as regiões não codificantes do DNA, a neutralidade genética, a regulação da expressão genética, a diferença de pressão de seleção sobre genes e códons específicos, o diploidismo e a consequente competição de genes por transcrição, o ambiente bioquímico e sua influência sobre a informação genética e, por fim, os diversos níveis de tradução desde o genótipo até o fenótipo.

Vários trabalhos têm sido realizados no sentido de representar alguns desses elementos, destacando-se os algoritmos meméticos e os algoritmos epigenéticos. Algoritmos meméticos integram principalmente a busca local (LS - *Local Search*) e métodos de otimização determinísticos ao processo evolucionário. Nesses algoritmos, uma etapa com busca local é normalmente realizada após a aplicação dos operadores de variação, e antes da seleção para a próxima geração. O algoritmo é chamado "memético" pois diz ser baseado no conceito de meme, definido por Dawkins [Dawkins 1976] como replicadores culturais. No caso nos algoritmos meméticos, eles representam informações não genéticas que são utilizadas para melhorar ou refinar o indivíduo [Krasnogor e Smith 2005, Smith 2007, Wanner et al. 2008, Ahn et al. 2010], ou

[7] esse processo é utilizado, por exemplo, pelos retrovírus, como o HIV.

seja, um algoritmo memético é um algoritmo híbrido com busca global e local. Uma das questões nesses algoritmos é a escolha de soluções na população para o refinamento local, que pode consumir processamento considerável. Abordagens mais simples intercalam a busca global (pelo algoritmo estocástico) com a busca local (pelo método de otimização determinístico) enquanto abordagens mais elaboradas realizam a busca local apenas nas regiões onde é mais provável que o mínimo global seja encontrado [Handoko, Kwoh e Ong 2010].

Os algoritmos epigenéticos representam o ambiente bioquímico que cerca o material genético e que, desse modo o influencia e manipula. O termo epigenético foi cunhado por C. Waddington em 1942 e significa "acima" ou "além" dos genes, e permite, por exemplo, que o mesmo código genético possua funcionalidades diferentes em diferentes tipos de células [Smith, Bolton e Nguyen 2010]. Atualmente, epigenético significa mudanças moleculares que regulam a função dos genes mas não envolvem mudanças de bases nucleicas no DNA, ou seja, mudanças genéticas. O ambiente bioquímico, ou intracelular, é indiretamente influenciado tanto pelo material genético quanto pelo ambiente externo no qual situa-se o indivíduo, pois ambos introduzem ou retiram, produzem, consomem ou transformam compostos nesse ambiente [Goldbarg, Bagi e Goldbarg 2008]. Mudanças epigenéticas incluem a metilação do DNA[8], modificações de RNA, de histonas e de outras proteínas.

Todavia, por vezes esses novos elementos não foram avaliados no contexto da otimização multiobjetivo usando indicadores e *benchmarks* conceituados e/ou a representação desses elementos é simplista, de modo que sua representação e avaliação de sua eficiência na otimização ainda precisam ser adequadamente verificados. Parte dessa avaliação é realizada nesta tese.

3.2 Algoritmos Evolucionários

3.2.1 Conceituação e Princípios Básicos

Algoritmos evolucionários são modelos computacionais de inspiração biológica, baseados na ideia de evolução pela seleção natural descrita na seção 3.1. Nesses modelos, cada variável v_i associada ao problema é mapeada

[8] a metilação é causada por enzimas e está associada ao silenciamento irreversível da expressão genética.

num *gene* g_i através de uma estratégia de codificação $\mathfrak{c}(v_i) \mapsto g_i$. Um *cromossomo* $\overrightarrow{CR} = [g_1, ..., g_c]$ é um vetor de genes relacionados de alguma forma, e normalmente associados à mesma função-objetivo. Um *indivíduo* $\overrightarrow{IND} = [\overrightarrow{CR}_1, ..., \overrightarrow{CR}_i]$ é um vetor de cromossomos, de forma que um indivíduo representa, de forma codificada, uma possível solução ao problema. Por fim, uma *população* $\mathscr{P} = \{\overrightarrow{IND}_1, ..., \overrightarrow{IND}_p\}$ é um conjunto de indivíduos de certa geração.

Depois de uma população inicial de α indivíduos ter sido gerada, o algoritmo evolui em direção a uma solução ótima num processo iterativo em que realiza a *seleção* dos indivíduos mais aptos (aproximações da solução ótima) e então sua *variação* (explorações de outras soluções) para compor uma nova geração da população, que será utilizada na próxima iteração. Esse processo básico é apresentado no algoritmo 3.1.

```
1  entrada: critério de parada, parâmetros de recombinação e mutação,
       tamanho da população, critérios de seleção
2  saída: pool de melhores indivíduos (A)
3  inicio
4     t = 0;
5     A_t = ∅
6     inicialize P_t;
7     faça
8        P'_t = recombinação P_t;
9        P''_t = mutação P'_t;
10       calcule aptidão φ de P''_t;
11       A_{t+1} = selecione (P''_t ∪ A_t);
12       P_{t+1} = selecione (P''_t ∪ A_t);
13       t = t+1;
14    enquanto critério de parada não atingido;
15 fim
```

Algoritmo 3.1: *Funcionamento básico de um algoritmo genético*

Esse mesmo processo também pode ser representado simplificadamente através da equação (3.1), onde $\mathscr{P}_t$ é a população no instante t (ou população da geração t), Sv é um operador de *seleção para reprodução* sobre a população, com parâmetros $\overrightarrow{\gamma}$, V é um operador de *variação* com parâmetros $\overrightarrow{\delta}$, Ss é um operador de *seleção para sobrevivência*, com parâmetros $\overrightarrow{\theta}$, que opera sobre a população atual, a população variada e sobre $\mathscr{A}_t$, que é o conjunto dos melhores indivíduos encontrados até a geração t, e $\mathscr{P}_{t+1}$ é a nova população, ou a nova geração da população [Purshouse e Fleming 2007].

$$\mathscr{P}_{t+1} = Ss\left(V(Sv(\mathscr{P}_t, \overrightarrow{\gamma}), \overrightarrow{\delta}), \mathscr{P}_t, \mathscr{A}_t, \overrightarrow{\theta}\right) \tag{3.1}$$

A seleção para reprodução utiliza uma *função de aptidão* $\phi(\overrightarrow{x}_{i,t}) : \overrightarrow{IND} \mapsto \Re$ que mapeia cada indivíduo da população atual a uma oportunidade de reprodução. Ela é distinta das *funções-objetivo* $\mathscr{F} = \{f_1, f_2, ..., f_M\}$ (ver eq. (2.1)), que demandam conhecimentos específicos do problema, e que mapeiam o *genótipo* de um indivíduo a um valor no domínio do problema sendo tratado. Esse valor representa o quão boa é essa solução. As *funções-objetivo*, portanto, definem o *fenótipo* do indivíduo, enquanto a função de aptidão utiliza os resultados das funções-objetivo como entrada para definir uma oportunidade de reprodução, e está intimamente relacionada ao operador de seleção.

Os indivíduos mais aptos à reprodução são selecionados de algum modo para formar casais de $\mu \in \overrightarrow{\gamma}$ indivíduos e gerar $\lambda \in \overrightarrow{\gamma}$ indivíduos da nova população $\mathscr{P}_{t+1}$. Seus genótipos são variados através de operadores como recombinação e mutação. Com base na população atual $\mathscr{P}_t$, nos novos indivíduos $Sv(\mathscr{P}_t, \overrightarrow{\gamma})$, e no conjunto dos melhores indivíduos até então ($\mathscr{A}_t$), os algoritmos utilizam heurísticas para manutenção da diversidade da população, controle da pressão de seleção para fazer a *seleção para sobrevivência Ss*, atualizando o conjunto elitista dos melhores indivíduos e a nova população. O processo segue iterativamente até que o critério de parada seja atingido.

A *pressão de seleção* é o grau no qual apenas os melhores indivíduos são escolhidos, e está diretamente relacionada à diferença entre a aptidão dos melhores e dos piores indivíduos. Pressões pequenas podem comprometer a convergência do algoritmo, enquanto pressões altas podem causar a convergência prematura, levando a mínimos locais. A *intensidade de seleção* ($\Upsilon_{\mathscr{P}}$) é a diferença da aptidão média da população ($\bar{\phi}$) após a seleção e antes da seleção, formalmente definida na equação (3.2). "De um ponto de vista global, uma das receitas para o melhoramento na evolução é intercalar períodos de forte seleção e períodos de relaxamento" [Dawkins 1986].

$$\begin{aligned} \Upsilon_{\mathscr{P}}(\mathscr{P}) &= \bar{\phi}(S(\mathscr{P})) - \bar{\phi}(\mathscr{P}) \\ \bar{\phi}(\mathscr{P}) &= \frac{1}{p}\sum_{i=1}^{p} \phi(\overrightarrow{IND_i}) \ , \ \overrightarrow{IND_i} \in \mathscr{P} \end{aligned} \tag{3.2}$$

A *diversidade da população* ($\Sigma_{\mathscr{P}}$) é uma medida da variação entre os indivíduos da mesma população, e é representada pelo desvio-padrão da função de aptidão da população, formalmente definida na equação (3.3). Dois são os

fatores que causam a diminuição da *diversidade da população*: pouca *pressão de seleção* e a *deriva genética*[9].

$$\Sigma_{\mathscr{P}}(\mathscr{P}) = \frac{1}{p}\sum_{i=1}^{p}\left(\phi(\overrightarrow{IND_i}) - \bar{\phi}(\mathscr{P})\right)^2, \overrightarrow{IND_i} \in \mathscr{P} \tag{3.3}$$

Outro conceito relacionado é o de *variação da população* ($\Delta_{\mathscr{P}}$), que é uma medida da diferença entre a aptidão média de uma população e de populações passadas, formalmente definido na equação (3.4). Ressalta-se que a intensidade de seleção, a diversidade da população e a variação da população referem-se ao *fenótipo* dos indivíduos, e não ao *genótipo*.

$$\Delta_{\mathscr{P}}(\mathscr{P}_t, \mathscr{P}_{t-1}) = \bar{\phi}(\mathscr{P}_t) - \bar{\phi}(\mathscr{P}_{t-1}) \tag{3.4}$$

O *critério de parada* define as condições nas quais assume-se que uma solução ótima global foi encontrada e o processamento pode ser encerrado. Esse é um ponto muito importante, embora nem sempre receba a importância devida. Os critérios mais usados, embora não sejam suficientemente bons [Ashlock 2006] são: (1) Parar se nenhuma mudança significativa ocorrer por um longo período; e (2) Parar depois de um tempo máximo admissível. Pode-se melhorar esses critérios de parada reexecutando o algoritmo com diferentes populações iniciais e observando a variação de seu comportamento. Também é possível substituir parte da população por uma nova população aleatória quando nenhuma melhora significativa ao longo do tempo for percebida.

Uma das abordagens propostas nesta tese é a agregação da exploração hierárquica ao critério de parada. A enorme quantidade de alternativas dificulta a exploração de todo o espaço de projeto simultaneamente, de modo que propõe-se que sejam fixados os genes associados a variáveis dos níveis hierárquicos inferiores ao que está sendo explorado num dado instante, e deixa-se a população convergir para a fronteira de Pareto. Quando um critério de parada é atingido, como por exemplo a população ter estagnado, baixa-se o nível de abstração sendo explorado e liberando a variação de outros genes para que sejam utilizadas soluções da fronteira de Pareto num nível de abstração como população inicial para a exploração de novos espaços de projeto no nível de abstração imediatamente

[9] *deriva genética* é um processo estocástico, atuante sobre as populações, que modifica a frequência dos alelos e a predominância de certas características na população. É mais frequente ocorrer em populações pequenas e as alterações induzidas podem não ser adaptativas.

inferior. Embora essa e outras abordagens possam ser facilmente realizadas, os experimentos apresentados nesta tese usam unicamente a quantidade máxima de gerações como critério de parada, para facilitar comparações com outras soluções.

3.2.2 Classificação e Características

O processo descrito na seção 3.2.1 é genérico, e vale para todos os tipos de algoritmos evolucionários. Algoritmos evolucionários podem ser classificados em algoritmos genéticos (GA - *Genetic Algorithm*), programação genética (GP - *Genetic programming*) e programação da expressão genética (GEP - *Gene Expression Programming*). Nos *algoritmos genéticos* (GA), os indivíduos são representados por strings simbólicas de comprimento fixo, os cromossomos. Na *programação genética* (GP), indivíduos são entidades não lineares de diferentes tamanhos e formas, as árvores de *parsing*. Na *programação da expressão genética* (GEP), indivíduos são codificados como strings simbólicas de tamanho fixo (cromossomos), que são posteriormente expressos (ou traduzidos) como entidades não lineares de diferentes tamanhos e formas, as árvores de expressão (ET - *Expression Tree*) [Ferreira 2001].

Esta classificação é análoga à apresentada por Miranda [Lee e El-Sharkwaki 2008], em que três formas de representação de indivíduos são possíveis: (1) *Fenótipo puro*, em que o indivíduo é representado pelas mesmas variáveis usadas pela função de aptidão; (2) *Fenótipo transformado*, em que o código que representa um indivíduo é composto de variáveis que, com uma transformação, podem ser mapeadas nas variáveis usadas na função de aptidão; e (3) *Genético*, em que o código que representa um indivíduo deve ser interpretado como um conjunto de instruções que permite construir o fenótipo de um indivíduo. O *genótipo* de um indivíduo é o vetor de símbolos que representam seus genes, enquanto seu *fenótipo* é o efeito das variáveis do problema em relação aos aspectos de interesse, representados pelos valores das funções-objetivo. Embora o termo "genético" seja utilizado em todos os casos, pela analogia biológica, ele deveria ser reservado aos sistemas nos quais as instruções para definição do fenótipo estão codificadas [Lee e El-Sharkwaki 2008]. Se as variáveis estão codificadas de forma praticamente direta nos GAs, a variação obtida pelos operadores é introduzida diretamente em nível do fenótipo. Os GPs, por outro lado, possuem operadores muito limitados e agem diretamente nas árvores de *parsing* [Castro e Zuben 2005] [Zielinski e Rutkowski 2006]. A GEP está livre

dessas restrições e age sobre o genótipo, como desejado.

Programação da Expressão Genética - GEP

Na GEP, indivíduos são entidades lineares que são traduzidos para uma estrutura hierárquica que, entre várias possibilidades, pode representar a expressão matemática de uma função-objetivo. Visando garantir sempre a formação de indivíduos válidos, os genes de um indivíduo são compostos de um cabeçalho h e uma cauda t. O cabeçalho possui símbolos que podem representar funções matemáticas ou terminais (operandos), enquanto a cauda contém apenas terminais [Ferreira 2001]. Para cada problema o tamanho do cabeçalho é determinado e o tamanho da cauda é dado pela equação (3.5), onde n é a quantidade de símbolos terminais.

$$t = h.(n-1)+1 \tag{3.5}$$

Em sua forma mais simples, com $h = 0$ e $n = 0$, GEP é igual a um GA [Ferreira 2002]. Num exemplo apresentado por Ferreira [Ferreira 2002], considere um conjunto de funções matemáticas $\mathscr{F} = \{Q, *, /, -, +\}$, onde Q representa a raiz quadrada, e um conjunto de terminais $\mathscr{T} = \{a, b\}$ (portanto, $n = 2$). Escolhendo $h = 15$, a cauda terá tamanho $t = 16$ e um gene terá tamanho $g = t + h = 31$. Um possível gene é apresentado na tabela 3.1. A equação expressa por esse gene é apresentada na equação (3.6), conforme o mecanismo de tradução proposto pela autora, que usa a linguagem Karva[10] para montar uma árvore de expressão (ET - *Expression Tree*), apresentada na figura 3.1(a). Formas alternativas de tradução da ET, como a P-GEP (P-GEP - *Prefix Gene Expression Programming*), proposta por Li et alli [Li et al. 2005], são menos disruptivas que Karva e também podem ser utilizadas, com impacto positivo em alguns operadores, como a recombinação. Para a otimização de sistemas complexos, também é mais eficaz o uso de cromossomos multigênicos, pois eles permitem a definição de estruturas hierárquicas e complexas, em que cada gene codifica um pequeno bloco dessa estrutura [Ferreira 2002].

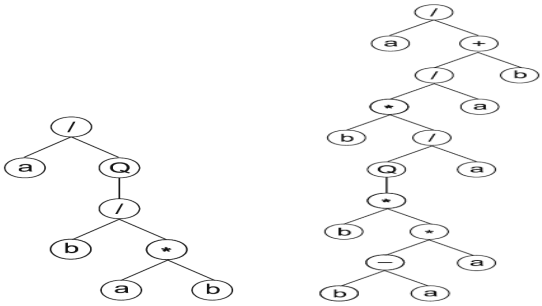
[10] Karva é uma forma de avaliar os símbolos do gene e gerar a árvore de expressão (ET), e que difere das formas postfix e prefix utilizadas na GP.

Tabela 3.1: *Exemplo de gene que representa expressão matemática*

	cabeçalho	cauda
posição:	000000000011111	1111122222222223
	012345678901234	5678901234567890
gene:	/aQ/b*ab/Qa*b*-	ababaababbabbbba

$$f(a,b) = \frac{a}{\sqrt{\frac{b}{a*b}}} \tag{3.6}$$

$$f(a,b) = \frac{a}{b * \frac{\sqrt{b*((b-a)*a)}}{a} + b} \tag{3.7}$$



(a) (b)

Figura 3.1: *Árvore de expressão traduzida a partir do gene da tabela 3.1. (a) Árvore de expressão traduzida a partir do gene da tabela 3.1. (b) Árvore de expressão traduzida a partir do gene da tabela 3.1 com uma mutação na posição 02, alterando o símbolo "Q" para "+".*
Fonte: [Ferreira 2002]

Um aspecto interessante na GEP é a existência de neutralidade genética, ou seja, a existência de símbolos presentes nos genes que podem não ser transcritos e, desta forma, não ter efeito no fenótipo. Trata-se de um efeito benéfico

gerado pela divisão dos genes em cabeçalho e cauda, e que também garantem que as expressões geradas serão válidas, ou seja, haverá operadores (terminais) para todas as funções existentes no cabeçalho [Castro e Zuben 2005]. Entretanto, não garante que não haverá operandos inconsistentes para certos operadores, como números negativos para a raiz quadrada. No gene da tabela 3.1 apenas os 7 primeiros símbolos são transcritos, e os demais são neutros. Entretanto, uma única mutação pode mudar significativamente a forma da árvore de expressão gerada e a quantidade de símbolos expressos e nulos. Ainda em relação ao exemplo apresentado por Ferreira [Ferreira 2002], uma única mutação no símbolo da posição 02, de "Q" para "$+$", muda a árvore de expressão para a apresentada na figura 3.1(b), que passa a expressar a equação apresentada na equação (3.7) e não mais a da equação (3.6).

A inclusão e tratamento de constantes [Ferreira 2003, Li et al. 2005, Ferreira 2006] permitem à GEP evoluir expressões matemáticas que representam uma equação qualquer, e são usadas nesta tese nesse sentido e também para representar dependências alternativas e mapeamentos entre componentes de software e hardware em sistemas embarcados. Em muitas áreas, uma função $f(\overrightarrow{x})$ de interesse, cuja expressão matemática seja desconhecida, mas da qual é possível obter seu valor para um conjunto de pontos, pode ter sua expressão aproximada por outra função $f'(\overrightarrow{x})$ gerada por técnicas como regressão ou interpolação, entre outras. Entretanto, essas técnicas limitam a forma da expressão aproximada, como por exemplo um polinômio de grau n, cuja forma geral é $f'(\overrightarrow{x}) = \sum_{i=0}^{n} a_i x_i^i$, onde os coeficientes a_i são definidos para que $f'(\overrightarrow{x})$ se ajuste a $f(\overrightarrow{x})$. Todavia, funções $f(\overrightarrow{x})$ não polinomiais, como as apresentadas nas equações (3.7) e (3.6), podem não ter boas aproximações com expressões na forma de um polinômio. O mesmo vale para outras formas fixas impostas por outras técnicas de ajuste. GEP permite que forma da expressão matemática seja livre e evolua para se ajustar melhor ao conjunto de pontos fornecidos.

Os elementos que constituem os genes da GEP não precisam ser decodificados apenas como operadores e operandos que formam uma ET para representar uma expressão matemática. Eles podem ser decodificados (ou traduzidos, em termos biológicos) de diferentes modos, dependendo do problema ou mesmo do tipo do gene. Ferreira apresenta exemplos em que os genes representam estruturas computacionais, como expressões booleanas, seleções (`IF-THEN-ELSE`) e atribuições, permitindo a evolução de algoritmos [Ferreira 2003] ou então representam elementos de um problema combinatorial, em que elementos precisam de um mapeamento um-para-um, como no problema do caixeiro viajante ou proble-

mas de escalonamento [Ferreira 2002]. Por sua versatilidade, EA do tipo GEP são usados nesta tese para representar os diversos aspectos da exploração do espaço de projeto, incluindo a estrutura de um sistema embarcado, diferentes tipos de variáveis envolvidas nos componentes, dependências alternativas e mapeamentos entre componentes, e também as próprias funções-objetivo usadas na otimização.

3.3 Questões Fundamentais para Otimização Multiobjetivo

3.3.1 Codificação

A *codificação* corresponde ao mapeamento entre as variáveis do problema e a representação computacional de um vetor de símbolos, que representam os genes de um indivíduo, denotada por $\mathfrak{c}(v_i) \mapsto g_i$. Esse mapeamento deve ser tal que, dado um indivíduo com determinada codificação genética, possa ser possível determinar uma única solução correspondente, através de uma função inversa $\mathfrak{m}(g_i) \mapsto v_i$, $\mathfrak{m} = \mathfrak{c}^{-1}$. A codificação é importante pois representa aspectos específicos do problema sendo tratado, de modo que o conhecimento do problema afeta a forma de codificação e seu desempenho. As principais questões dizem respeito à forma como a codificação é feita, à interação entre genes e à forma como codificações inválidas são tratadas, e é talvez a característica mais importante de um algoritmo evolucionário [Bonissone e Subbu 2007].

As duas principais abordagens para a forma de codificação são a *binária* e a de *variáveis reais*. Na *codificação binária* as variáveis são codificadas num alfabeto binário de tamanho fixo e suficiente para armazenar os valores possíveis da variável. A forma de representação binária das soluções também afeta operadores de variação, como mutação e recombinação, e pode gerar comportamentos bizarros na representação de números reais [Ashlock 2006]. Por sua vez, a posição de um gene no cromossomo pode afetar o operador de recombinação, de forma que é importante agrupar variáveis relacionadas ou escolher cuidadosamente o funcionamento desse operador. Embora já tenha sido muito usada, atualmente a *codificação por variáveis reais* tem sido mais adotada para problemas mais complexos [Ashlock 2006, Lee e El-Sharkwaki 2008]. Uma forma alternativa está presente basicamente na GEP, já que as variáveis não são codificadas diretamente, mas sim instruções que permitem determinar o valor de uma variá-

vel após uma tradução (no sentido biológico) adequada. Nesta tese uso codificação por variáveis reais para muitas variáveis, mas alguns elementos do projeto de sistemas embarcados, como mapeamento de componentes de software a componentes de hardware, dependências alternativas entre componentes e variáveis discretas são codificados como sugerido pela GEP para tratamento de constantes.

A *epistasia* é a interação entre genes, causada quando um gene é modificado por um ou muitos outros genes, chamados genes modificadores. Uma das suposições feitas nos EA é que os valores de diferentes genes são independentes entre si [Chakraborty, Mitra e Roychoudhury 2006, Lee e El-Sharkwaki 2008], ou seja, que não ocorre epistasia. Gras [Gras 2008] demonstrou que EA são incapazes de resolver mesmo problemas simples com epistasia quando não há informação sobre as interações entre os genes. Uma vez que interações entre variáveis são comuns no projeto de sistemas embarcados, técnicas adequadas para tratamento de epistasia são necessárias durante a codificação. Uma das abordagens utilizadas nesta tese para tratar a epistasia é a inclusão de unidades de informação mais elementares que os genes: os códons. Por serem a unidade básica de informação biológica, eles correspondem mais adequadamente às variáveis dos problemas. Genes podem ter um único ou muitos códons, e variáveis relacionadas podem ser modeladas como códons de um único gene.

Para qualquer codificação pode haver mais códigos num gene do que valores possíveis para a respectiva variável, o que gera códigos inválidos que precisam ser tratados. Para isso, pode-se usar cinco abordagens principais: (1) Rejeitar os indivíduos com codificação inválida da população; (2) Fazer o mapeamento dos códigos excedentes a soluções válidas; (3) Alterar a codificação inválida para uma codificação válida; (4) Garantir a priori que toda codificação seja válida; e (5) Manter indivíduos inválidos, mas penalizá-los.

As quatro primeiras abordagens nunca geram soluções inválidas, sendo que a forma como códigos redundantes são gerados na segunda abordagem também influencia a probabilidade de representação dos valores da variável em questão e, consequentemente, a exploração de novas soluções. Para sistemas complexos, as soluções ótimas podem estar próximas da fronteira entre soluções viáveis e inviáveis, o que costuma ser verdade particularmente no projeto de sistemas embarcados. Desse modo, impedir a representação de soluções inviáveis pode dificultar encontrar as soluções ótimas. A última abordagem, por outro lado, permite soluções inválidas, mas as torna menos aptas (seção 3.3.2).

Vale ressaltar que os EA podem desempenhar diferentes papéis no projeto

de um sistema embarcado. Em sua utilização mais comum, os indivíduos representam soluções para o problema, que são avaliados por funções-objetivo e então selecionados conforme uma função de aptidão. Em outros casos, entretanto, as próprias expressões matemáticas das funções-objetivo podem ser desconhecidas e EAs podem ser usados para descobri-las [Ferreira 2002, Ferreira 2003]. Nestes casos, soluções ou indivíduos inválidos correspondem a expressões matemáticas inválidas, o que é indesejável. Nesse caso, abordagens em que indivíduos que representam expressões inválidas nunca são gerados são preferíveis. Em particular, a quarta abordagem apresentada, em que a validade da codificação é garantida a priori, possui atrativos interessantes nesses casos. A GEP apresenta essa possibilidade de forma elegante, conforme foi apresentado na seção 3.2.2.

Nesta tese utilizo a quinta abordagem para representação do projeto de sistema embarcado. Porém, para alguns aspectos do projeto, como mapeamento entre software e hardware e dependências alternativas entre componentes, soluções inválidas não são úteis. Nesses casos específicos, utilizo algumas ideias da GEP para tratamento de constantes e o auxílio de mecanismos de tradução do genótipo que garantem a priori que mapeamentos e dependências sempre serão válidos.

3.3.2 Seleção Ambiental

Os *operadores de seleção* são usados para escolher, dentre os indivíduos da população atual, aqueles que serão usados para gerar a próxima geração da população. Nos MOEA, a seleção visa escolher indivíduos que pertençam ou tenham maior probabilidade de pertencer ao conjunto de Pareto. Há duas classes principais de operadores de seleção: proporcionais e ordinais. Os *operadores proporcionais* são operadores de seleção em que a probabilidade de um indivíduo ser selecionado para reprodução é proporcional à sua aptidão. Um problema nessa classe de operadores é que a *intensidade de seleção* é pequena, já que o melhor e o pior indivíduos determinam a abrangência da aptidão [Lee e El-Sharkwaki 2008]. A diferença de aptidão entre indivíduos pode causar a predominância de poucos deles, causando uma convergência prematura. Com isso, operadores proporcionais usualmente requerem transformações na função de aptidão e são sensíveis à distribuição da população. Em contrapartida, os *operadores ordinais* são operadores de seleção que escolhem indivíduos com base em sua ordenação relativa na população, e não diretamente em sua aptidão. Essa ordenação normalmente

relaciona-se à quantidade de soluções dominadas, usando alguma forma de preferência das soluções ($\preceq_p, \preceq_d, \preceq_v, \preceq_l$, etc) . Com isso, não apresentam os problemas dos operadores proporcionais e a influência do melhor e o pior indivíduos é desprezível, não importando quão semelhantes ou diferentes eles sejam. Em praticamente todas as formas de seleção para MOO os operadores ordinais são usados.

A seleção também pode tratar de penalidades para manter um grau de convergência adequado. Penalidades devem ser aplicadas caso soluções inviáveis possam ser codificadas, diminuindo sua aptidão. Para tanto, a função de aptidão deve ser ajustada por fatores de penalidade[11]. Fatores de penalidade muito pequenos podem retardar a convergência do algoritmo (pelo aumento da quantidade de soluções inviáveis), enquanto fatores muito altos podem rejeitar soluções que são inviáveis, mesmo que possivelmente estejam muito próximas da solução ótima. Chakraborty et alli [Chakraborty, Mitra e Roychoudhury 2006] citam que há problemas no domínio de sistemas embarcados em que uma grande proporção do espaço de projeto é inviável, e que uma grande parte da população pode representar soluções inviáveis em poucas gerações. Eles sugerem uma abordagem híbrida (baseada em *branch-and-bound*) para reparar soluções inviáveis. Assim, a convergência do algoritmo e a proporção de soluções inviáveis representadas na população podem ser ajustadas tanto pelo grau de penalidade aplicado quanto pelo uso de técnicas de reparação. O uso dessas estratégias pode ser alterado ao longo das gerações e podem também estar codificado no próprio indivíduo, permitindo sua adaptação. A última abordagem citada é utilizada nesta tese.

Os indivíduos selecionados podem ser alterados posteriormente por *operadores de variação*, conforme é apresentado na seção 3.3.3. Decorre deste fato que as melhores soluções já encontradas podem desaparecer da população. Para evitar isso, a seleção deve também promover alguma forma de *elitismo*, que é uma garantia da inclusão das melhores soluções da população atual na próxima população, preservando as melhores soluções já encontradas. Várias pesquisas comprovam a importância do elitismo na estratégia de seleção [Zitzler 1999, Lee e El-Sharkwaki 2008].

Outra função importante da seleção é manter a *diversidade da população*. Uma das maneiras de fazer isso é incluindo *nichos*, que são sub-populações razoavelmente estáveis, análogas a povoados geograficamente isolados. *Nichos* po-

[11] um para cada variável violada, já que penalidades com base na quantidade de violações não funcionam bem [Lee e El-Sharkwaki 2008].

dem ser baseados em compartilhamento de aptidão, esquemas de multidão, reprodução restrita ou sub-populações [Lee e El-Sharkwaki 2008]. Todos eles buscam encontrar indivíduos com genótipo ou fenótipo *similares* para formar um nicho. O *compartilhamento de aptidão* diminui a aptidão de cada indivíduo proporcionalmente à quantidade de indivíduos similares na população. *Esquemas de multidão* definem vizinhanças com base na similaridade dos indivíduos. Sempre que um novo indivíduo é gerado, um dos indivíduos de sua vizinhança desaparece. Na *reprodução restrita*, apenas indivíduos similares podem ser cruzar entre si para criar um novo indivíduo. Por fim, é possível criar sub-populações totalmente distintas, que evoluem de forma independente (possivelmente em computadores remotos), e que ocasionalmente trocam indivíduos entre si, num processo conhecido como *migração*.

O indicador de hipervolume tem sido utilizado como indicador de qualidade de um otimizador e representa, num único valor numérico, a proximidade das soluções da fronteira de Pareto e também sua distribuição. Algumas estratégias, como as propostas por [Bader e Zitzler 2008, Bradstreet, Barone e While 2009, Li et al. 2009], isolam a contribuição de soluções individuais sobre o indicador de hipervolume. Desse modo, pode-se afirmar que as melhores soluções são aquelas que têm maior contribuição sobre o valor final do indicador de hipervolume. Knowles e Corne [Knowles e Corne 2002] foram os primeiros a propor a integração do indicador de hipervolume no processo de otimização. Entre as vantagens de seu uso como operador de seleção está um maior ordenamento das soluções, o que diminui a quantidade de soluções incomparáveis (equação 2.9), que são difíceis de tratar em qualquer otimizador [Bader 2009]. Nesta tese foi desenvolvido o arcabouço para utilização do hipervolume como operador de seleção, mas os experimentos realizados utilizaram o NSGA2, por facilitar a comparação com outras soluções já publicadas.

3.3.3 Variação Aleatória

Os *operadores de variação* definem o modo no qual indivíduos-pais geram indivíduos-filhos (*offsprings*) e como alterações genômicas aleatórias são aplicadas. As consequências fenotípicas são avaliadas por mudanças nas aptidões desses indivíduos, no nível do problema sendo tratado. Dentre os operadores de variação mais conhecidos, estão a mutação e a recombinação. Contudo, dezenas

de operadores e variações são encontrados na literatura[12], de modo que é importante compreender sua importância e utilidade na otimização de problemas [Ferreira 2002].

O operador de mutação é aplicado sobre um único indivíduo $\overrightarrow{IND_i}$, e é, de longe, o operador de variação mais importante [Ferreira 2002]. A figura 3.2(a) representa o efeito do operador de mutação, deslocando a solução para outro ponto do *espaço de decisão*. Quando a codificação é *binária*, costuma-se inverter bits aleatoriamente, com certa probabilidade, conhecida como taxa de mutação $\zeta \in \overrightarrow{\delta}$. Para codificação por *variáveis reais*, pode haver substituição aleatória do valor de uma variável, ou então somar ou multiplicar o valor de uma variável por uma quantidade aleatória. Neste último caso, essa quantidade aleatória pode seguir diferentes distribuições de probabilidade, como uma uniforme ou uma normal. Neste caso, o operador de mutação pode ser definido pela equação (3.8)

$$\overrightarrow{\widetilde{IND}} = \begin{cases} \bigcup_{i=1}^{m-1} g_i \bigcup (g_m + \xi) \bigcup_{i=m+1}^{c} g_j & , U(0,1) \leq \zeta \\ \overrightarrow{IND} & , U(0,1) > \zeta \end{cases} \tag{3.8}$$

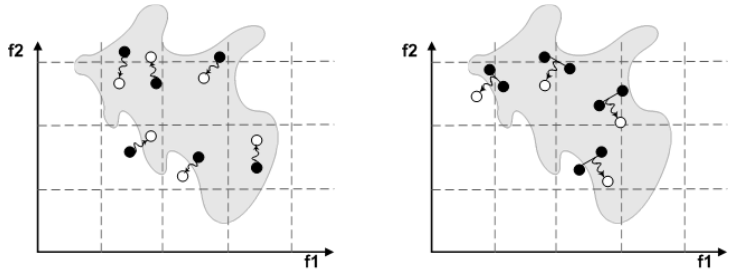
$$\xi = \begin{cases} b(U(0,1) - 0,5) & , \text{se uniforme} \\ \sigma N(0,1) & , \text{se normal} \\ e^{\sigma N(0,1)} & , \text{se lognormal} \\ \ldots & \end{cases} \tag{3.9}$$

onde $\overrightarrow{\widetilde{IND}}$ é o novo indivíduo mutado, g_m é o gene sendo mutado, $U(0,1)$ é uma função que retorna uma variável uniformemente distribuída entre zero e um[13], ξ é a distribuição do operador de mutação, $N(0,1)$ é uma função que retorna uma variável aleatória com distribuição normal-padrão, e b e σ são parâmetros das distribuições uniforme e normal, respectivamente. Para genes que representam variáveis discretas a distribuição de mutação pode seguir uma Poisson [Lee e El-Sharkwaki 2008]. Substituições totalmente aleatórias costumam levar a mutações expressivas, e são mais recomendadas nas gerações iniciais; Alterações das variáveis usando uma distribuição normal têm maior probabilidade de gerar pequenas alterações no fenótipo do indivíduo, e são mais recomendadas em gerações mais tardias, quando os indivíduos já estão próximos de soluções ótimas e deseja-se apenas fazer pequenos ajustes para encontrá-las. Com base nisso,

[12] inversão, transposição, inserção, remoção, rotação, etc.

[13] ou seja, um GNA (GNA - *Gerador de Números Aleatórios*).

pesquisas recentes têm tentado usar taxas e distribuições distintas para indivíduo e ainda estabelecer correlações entre as mutações [Tran 2006].



(a) Efeito do operador de mutação

(b) Efeito do operador de recombinação

Figura 3.2: *Efeito básico dos operadores de mutação e de recombinação*

Ashlock [Ashlock 2006] sugere taxas de mutação variáveis, de tal modo que algumas mutações façam uma busca abrangente e outras uma busca local. Desse modo, a mutação também pode ser ajustada de forma análoga às técnicas de *Hill-climbing* ou de *Particle Swarm*. O operador de mutação também torna-se mais importante à medida que as gerações de populações se passam, pois a maioria dos indivíduos apresenta qualidade similar [Lee e El-Sharkwaki 2008]. Zhang et alli [Zhang et al. 2009] propõem um método para que as taxas de mutação e de recombinação sejam adaptadas para cada indivíduo conforme a aptidão da população se torna mais uniforme ou distinta, ou seja, mude sua diversidade. A ideia é aumentar essas taxas quando a população se torne menos diversificada e diminuir as taxas quando aumenta a diversidade. Para isso utilizam uma equação que ajusta as taxas para cada indivíduo proporcionalmente à diferença entre sua aptidão e a aptidão média da população. Garantir robustez no ajuste de parâmetros de operadores genéticos, como o da mutação, ainda é um tema que requer alguma atenção [Purshouse e Fleming 2007].

Por esses motivos, este autor considera que a taxa de mutação e a distribuição da mutação devem ser variáveis, e preferencialmente adaptáveis, de forma que podem estar codificadas nos genes dos indivíduos. Nesta tese, a taxa de mutação pode ser distinta para cada códon num gene, melhor representando a pressão de seleção que ocorre nos códons que participam de sítios ativos numa proteína, e também pode ser distinta para cada gene num cromossomo, represen-

tando também a pressão sobre os genes com maior impacto na aptidão dos indivíduos. Entretanto, ao contrário de outras pesquisas, não foi incluído qualquer "design inteligente" direcionando a evolução. Proponho que o próprio mecanismo de evolução se encarregará de manter baixas as taxas de mutação nos elementos com maior pressão. Uso uma abordagem análoga também para o operador de recombinação.

O *operador de recombinação* define como o genótipo de indivíduos-pais é recombinado para formar o genótipo de novos indivíduos-filhos. A figura 3.2(b) representa o efeito do operador de recombinação, também deslocando a solução para outro ponto do *espaço de decisão*. Há muitas variações do operador de recombinação, e a escolha da recombinação adequada (ou a criação de um nova) é dependente do problema. Na recombinação, os indivíduos-pais são alinhados e o material genético de ambos é recombinado para formar os indivíduos-filhos, seguindo algumas abordagens diferentes. As abordagens mais populares são: (1) *n-pontos*, em que n pontos de cruzamento são aleatoriamente selecionados (tipicamente, um ou dois pontos). O genoma é cortado em $n+1$ partes e então recombinado trocando as partes de cada cromossomo-pai; e (2) *uniforme*, em que dois novos genomas (A' e B') são recombinados a partir de A e B por sucessivamente decidir se o gene A_n será incluído no indivíduo-filho A' e o gene B_n em B' ou então se o gene A_n será incluído em B' e B_n em A'. Esta abordagem também é chamada de recombinação de genes, e é um caso particular da recombinação *em blocos*, quando um bloco tem exatamente o tamanho de um gene. Ferreira [Ferreira 2002] cita que a recombinação de 2-pontos é a que mais causa rupturas, mas que também é a forma mais eficiente de recombinação, e que quando uma população evolui apenas pela recombinação, ela tende a convergir rapidamente, antes de encontrar uma solução ótima, de modo que a recombinação nunca deve ser usada sozinha, como única forma de variação genética. Conforme foi mencionado na seção 3.3.1, o operador de recombinação utilizado nesta tese alinha os cromossomos logicamente, já que a posição física dos genes é considerada inadequada para tratar a epistasia. Vale lembrar que também são representados genes alelos que competem por sua expressão. Nesses casos, o indivíduo-filho pode ter genes alelos de ambos os indivíduos-pais, como ocorre na realidade biológica.

3.3.4 Parametrização e Auto-Adaptação

Os parâmetros básicos de um algoritmo evolucionário incluem a quantidade máxima de gerações (N), o tamanho da população (α), o critério de parada, o número de pais por geração (λ), o número de filhos por geração (μ), e a taxa de aplicação de cada operador genético de variação ($\overrightarrow{\delta}$), entre outros. De modo geral, esses parâmetros são escolhidos empiricamente pelo projetista, o que pode comprometer os resultados da otimização. Ao longo da última década tem havido várias propostas no sentido de permitir que esses parâmetros sejam alterados ao longo das otimizações, e em algumas delas, com base no próprio mecanismo evolucionário de otimização.

Zhang et alli [Zhang et al. 2009] propõem um método para que as taxas de mutação e de recombinação sejam adaptadas para cada indivíduo conforme a aptidão da população se torna mais uniforme ou distinta, ou seja, mude sua diversidade. A ideia é aumentar essas taxas quando a população se torne menos diversificada e diminuir as taxas quando aumenta a diversidade. Para isso utilizam uma equação que ajusta as taxas para cada indivíduo proporcionalmente à diferença entre sua aptidão e a aptidão média da população.As taxas de mutação ζ e de recombinação φ propostas são formalizadas nas equações (3.10) e (3.11), respectivamente.

$$\zeta = \begin{cases} \zeta - \frac{\zeta_1 - \zeta_2}{g_{max} - g_{avg}} (g(\overrightarrow{IND}) - g_{avg}) & , g(\overrightarrow{IND}) > g_{avg} \\ \zeta & , f < g(\overrightarrow{IND}) \leq g_{avg} \end{cases} \tag{3.10}$$

$$\varphi = \begin{cases} \varphi - \frac{\varphi_1 - \varphi_2}{g_{max} - g_{avg}} (g(\overrightarrow{IND}) - g_{avg}) & , g(\overrightarrow{IND}) > g_{avg} \\ \varphi & , f < g(\overrightarrow{IND}) \leq g_{avg} \end{cases} \tag{3.11}$$

CAPÍTULO 4

EXPLORAÇÃO DO ESPAÇO DE PROJETO EM SISTEMAS EMBARCADOS

Um projeto é aquilo que o projetista tem quando acaba o tempo e o dinheiro.
—JAMES POOLE (The Fifth 637 Best Things Anybody Ever Said, 1993)

Este capítulo sobre exploração do espaço de projeto de sistemas embarcados está organizado em duas seções principais. A seção 4.1 apresenta uma visão geral de alguns fluxos de projeto e características gerais de metodologias de projeto. Duas metodologias principais são destacadas: PbD e ADESD. A seção 4.2 trata especificamente da exploração do espaço de projeto, e apresenta conceitos principais, técnicas utilizadas e exemplos de pesquisas relacionadas, focando em suas limitações e pontos interessantes que poderiam ser usados na exploração de projetos com a metodologia ADESD, usada nesta tese.

4.1 Projeto de Sistemas Embarcados

Sistemas Embarcados são dispositivos que incluem um sistema programável, mas que não são, por si, computadores de propósito geral [Wolf 2001]. Também podem ser entendidos como subsistemas eletrônicos de aplicação específica que são usados em sistemas maiores, como aparelhos domésticos, dispositivos médicos ou automobilísticos [Jerraya e Wolf 2005]. Com a disseminação dos sistemas embarcados, o avanço das tecnologias que os suportam e o consequente aumento de complexidade das aplicações embarcadas, cresceu também a necessidade de metodologias e técnicas adequadas ao seu desenvolvimento.

Durante as duas últimas décadas, o desenvolvimento baseado em componentes (CbD - *Component-based Development*), aliado a uma boa engenharia de software, tornou-se a principal abordagem ao desenvolvimento de software reusável para sistemas embarcados. Porém, a reusabilidade não emerge

pura e simplesmente dos componentes. Componentes devem ser projetados para serem reusáveis, ou não o serão. Assim, pesquisadores desenvolveram diversas metodologias baseadas no uso de componentes, ou de elementos de mais alto nível (como modelos, por exemplo) como entidades reusáveis, e passaram a permitir a integração dessas entidades no fluxo de projeto de sistemas embarcados. Um dos desafios que tem sido tratados é apresentação, para a aplicação, da mesma interface de programação (API - *Application Programming Interface*) para diferentes arquiteturas e também os mesmos componentes, de forma que o desenvolvimento da aplicação possa ser realizado de forma transparente à plataforma arquitetural utilizada. Sistemas operacionais de propósito geral e hardware de propósito geral normalmente não atendem adequadamente aplicações embarcadas, o que exige a análise do domínio de aplicações embarcadas para a determinação eficiente dos componentes que compõem o sistema, incluindo sua API. Outro problema está relacionado à granularidade dos componentes reusáveis. Uma granularidade muito fina produz componentes que não são auto-contidos e possuem muito acoplamento com outros componentes. Uma granularidade muito grossa produz componentes que fornecem mais serviços do que pode ser necessário à aplicação. Questões como essas devem ser abordadas adequadamente pelas metodologias de projeto.

A abordagem mais promissora no projeto de hardware também é desenvolver um sistema como uma composição de componentes reusáveis, usualmente denominados *IP cores*, ou simplesmente IP (IP - *Intellectual Properties*), uma vez que reduz a complexidade e tempo de projeto [Sangiovanni-Vincentelli e Martin 2001, Wang, Kodase e Shin 2002]. *IP cores* são normalmente descritos numa linguagem de descrição de hardware (HDL - *Hardware Description Language*) e interconectados de modo transparente em barramentos em chip (OCB - *On-Chip-Bus*) com interface padronizada, e diferentes barramentos podem ser interconectados utilizando componentes de hardware do tipo *bridges* ou *wrappers*. Mais recentemente, a interconexão de IP através de redes em chip (NoC - *Network-on-Chip*) tem sido foco de muitas pesquisas [Palma et al. 2005, Bjerregaard e Mahadevan 2006]. Além disso, também tem sido foco de muita pesquisa a síntese automática de componentes de hardware a partir de descrições de mais alto nível, como C++ [Micheli 1999, Kuhn e Rosenstiel 2000] ou SystemC [Initiative 2011].

Deve-se observar que a abordagem de *IP cores* pressupõe que o suporte de hardware será constituído por um dispositivo de lógica programável (PLD - *Programmable Logic Device*), como um FPGA (FPGA - *Field Programmable*

Gate Array), pelo menos temporariamente para prototipação e posterior manufatura de um ASIC (ASIC - *Application Specific Integrated Circuit*), o que nem sempre pode ser verdadeiro para um sistema embarcado. Nos sistemas que não são constituídos por PLDs ou chips desenvolvidos especificamente, os componentes reusáveis de hardware ainda assumem a forma de componentes físicos pré-fabricados (os componentes e chips eletrônicos tradicionais). Ressalta-se que esse tipo de sistema é comum, que a integração desses componentes físicos passa por questões similares às dos IP (aliás, foram os tradicionais componentes físicos de hardware que inspiraram os IP sintetizáveis), e que ferramentas de projeto deveriam considerar também essa classe de sistemas embarcados, não ficando restritas às FPGAs e ASICs. Apesar de não permitirem o desenvolvimento de sistemas embarcados como SoCs e nem o particionamento hardware/software, componentes físicos de hardware são reusáveis, confiáveis, têm interface padronizada e podem ainda constituir a melhor solução para vários sistemas embarcados. Assim, deve-se distinguir o suporte de hardware físico do suporte de hardware sintetizável. Abordagens mais recentes representam também a parte mecânica de um sistema embarcado, incluindo tanto a parte lógica (cyber) quanto a parte física (*physical*) de um sistema, formando um CPS (CPS - *Cyber Physical System*).

Há ainda grandes problemas a serem resolvidos em relação a metodologias e ferramentas para auxílio ao projetista e à automatização do projeto de sistemas embarcados, que dizem respeito aos suportes de software e de hardware para a aplicação-alvo. Diversas propostas foram realizadas, principalmente durante as duas últimas décadas, com muitos avanços significativos em diferentes aspectos do projeto. Neste capítulo, o foco é dado a fluxos de projeto de sistemas embarcados e em duas principais metodologias de projeto.

4.1.1 Fluxo de Projeto

Em sistemas baseados em componentes, o fluxo de projeto pode ser dividido em duas fases distintas: Fluxo de projeto da infra-estrutura e fluxo de projeto da aplicação. O *fluxo de projeto da infra-estrutura* corresponde às etapas necessárias para produzir os artefatos reusáveis que serão utilizados no desenvolvimento de (várias) futuras aplicações. Esses artefatos reusáveis normalmente correspondem a componentes de hardware e software validados, modelos e depuradores que populam um repositório [Pietro-Diaz 1990]. Para serem efetivamente reusáveis, eles devem ser pensados no contexto de diferentes aplicações, sendo o

produto de técnicas adequadas a esta finalidade, como engenharia de domínio (DE - *Domain Engineering*). A engenharia de domínio identifica e documenta o que há em comum entre várias aplicações de um mesmo domínio, permitindo um curto ciclo de desenvolvimento para futuras aplicações [Pietro-Diaz 1990]. Esse fluxo de projeto corresponde à análise do domínio, arquitetura do domínio, e desenvolvimento dos artefatos [Comer 1990]. Sem a aplicação criteriosa da engenharia de domínio, dificilmente o desenvolvedor da aplicação estará livre de alterar a implementação de componentes da infra-estrutura cada vez que desenvolver novas aplicações, e essas alterações à nova aplicação usualmente desconsideram as aplicações passadas, de modo que o componente alterado já não serve mais a elas. Assim, na opinião deste autor, o isolamento do fluxo de projeto da infra-estrutura do fluxo de projeto da aplicação, e a aplicação criteriosa de engenharia de domínio como base da criação de componentes reusáveis é indispensável ao projeto de sistemas embarcados eficientes, embora não seja suficiente.

O *fluxo de projeto da aplicação*, e tradicionalmente de aplicações de software para sistemas de uso geral, é estabelecido pela corrente de engenharia de software predominante. O projeto de aplicações orientado a objeto (OOD - *Object Oriented Design*) é um dos mais utilizados atualmente, e é composto principalmente pelas etapas de modelagem, projeto, implementação e testes. Mais recentemente, essa comunidade tem adotado o fluxo de projeto definido pela MDA (MDA - *Model Driven Architecture*), que foi estabelecido pela OMG (OMG - *Object Management Group*) [Group 2011]. O uso de modelos no cerne do processo de desenvolvimento tem merecido muito destaque, e o uso de metamodelos e a exploração de meta-informação facilita muitas tarefas do projeto de sistemas, como análise, verificação, síntese e geração de testes [Sangiovanni-Vincentelli et al. 2009]. O fluxo de projeto de aplicações embarcadas que integram hardware e software parece ter seguido caminho diferente, utilizando propostas baseadas no projeto conjunto de software e hardware (hardware/software co-design).

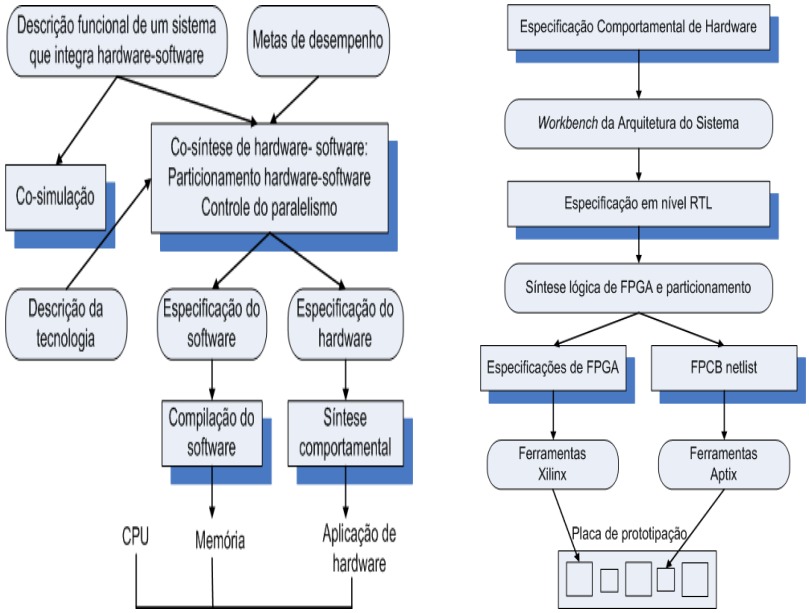
Thomas, Adams, e Schmit [Thomas, Adams e Schmit 1993] apresentaram, ainda em 1993, um fluxo de projeto que representa muitas soluções desenvolvidas nas duas últimas décadas. Esse fluxo já incluía particionamento hardware-software, síntese comportamental, compilação de software, e um ambiente de teste consistindo de FPGAs, e é ilustrado na figura 4.1(a). Duas tarefas eram consideradas principais: co-simulação e co-síntese. O particionamento hardware/software era uma das primeiras etapas, e visava satisfazer as restrições de projeto, escolhendo o melhor particionamento possível. Nesse caso, o

particionamento correspondia a uma exploração do espaço de projeto mais limitada. De forma geral, o fluxo usual de projeto conjunto de software e hardware inclui: especificação do sistema, particionamento hardware-software, co-síntese e co-simulação [Mei, Schaumont e Vernalde 2000]. Nas duas últimas décadas houve inúmeras contribuições nos métodos e nas técnicas usadas em hw/sw co-design, incluindo [Ernst et al. 1993, Balarin et al. 1997, Ernst e Jerraya 2000, Thiele et al. 2001, Jerraya e Wolf 2005], e refinamentos nas metodologias ocorreram ao longo desse período. Em 2004 o fluxo de projeto já incluía: análise do sistema, especificação, particionamento hardware/software, projeto preliminar, projeto detalhado, implementação de componentes, teste de componentes, integração de componentes, teste de módulos, integração do sistema, e testes do sistema para entrega [Muller-Glaser et al. 2004] Entretanto, fluxos como este misturam o fluxo da infra-estrutura com o fluxo da aplicação, e não são muito adequados para o desenvolvimento baseado em componentes, pois não consideram, previamente, o domínio do problema e aplicações futuras para o desenvolvimento da infra-estrutura (quem incluem os componentes reusáveis).

Nesta tese, assumo que os componentes que irão constituir o software de suporte à aplicação e o suporte de hardware já estão prontos. Esse não é um limitante se os fluxos forem distintos e a infraestrutura reusável já estiver disponível quando uma aplicação embarcada for projetada.

4.1.2 Projeto Baseado em Plataforma

A metodologia de Projeto Baseado em Plataforma (PbD - *Platform-based Design*) foi proposta inicialmente por Ferrari e Sangiovanni-Vincentelli em 1999 [Ferrari e Sangiovanni-Vincetelli 1999], e tem sido amplamente usada por facilitar o reuso de componentes de hardware, baseando o projeto de software numa plataforma composta por componentes de hardware pré-concebidos e parametrizáveis, auxiliando no ajuste dos parâmetros da plataforma para obter o melhor desempenho ao menor custo [Catania et al. 2009]. Os princípios básicos da PbD incluem iniciar o projeto no maior nível de abstração, abstrair detalhes desnecessários, incluir apenas os parâmetros importantes da implementação num modelo abstrato, limitando a exploração do espaço de projeto a um conjunto de componentes disponíveis, realizando o projeto a partir deles [Sangiovanni-Vincentelli 2007]. Dado que o processo de projeto tem se tornado mais complexo, criar um novo projeto “a partir do zero” é quase im-



(a) Fluxo de projeto da metodologia (b) Fluxo de síntese de hardware

Figura 4.1: *Metodologia de projeto conjunto de hardware/software. O fluxo de projeto inclui particionamento hardware-software, síntese comportamental, compilação de software, e um ambiente de teste consistindo de FPGAs. Duas tarefas eram consideradas principais: co-simulação e co-síntese.*
Fonte: [Thomas, Adams e Schmit 1993]

praticável, e muitos projetistas de sistemas embarcados adotaram a abordagem baseada em plataforma para lidar com esse problema [Horowitz et al. 2003, Al Rayahi e Khalid 2009, Tung et al. 2010]. A PbD assume que projetistas de sistemas embarcados irão usar cada vez mais software e que, portanto, metodologias devem focar no reuso de hardware. Isso implica que os componentes principais da micro-arquitetura básica (CPU, E/S, memória) permanecem os mesmos, apenas com um certo grau de parametrização (frequência de CPU, tamanho de memória, etc) [Keutzer et al. 2000], e que as plataformas de hardware devem ser abstraídas num nível mais alto para serem tratadas pelo software. PbD assume também que a plataforma será reutilizada para diversas aplicações ou fu-

turas evoluções de uma aplicação e, portanto, não será projetada apenas para um sistema específico que se deseja desenvolver. Nesse sentido, PbD costuma levar a plataformas no qual certo grau de recursos desnecessários estão presentes para um produto específico [Keutzer et al. 2000]. Cabe ressaltar que PbD define *plataforma* como "uma abstração que cobre vários possíveis refinamentos em níveis mais baixos. Cada plataforma provê uma perspectiva da qual mapeia-se camadas mais abstratas numa plataforma, e da qual definem-se classes de nível mais baixo de abstração que as plataformas implicam" [Sangiovanni-vincentelli e Martin 2001].

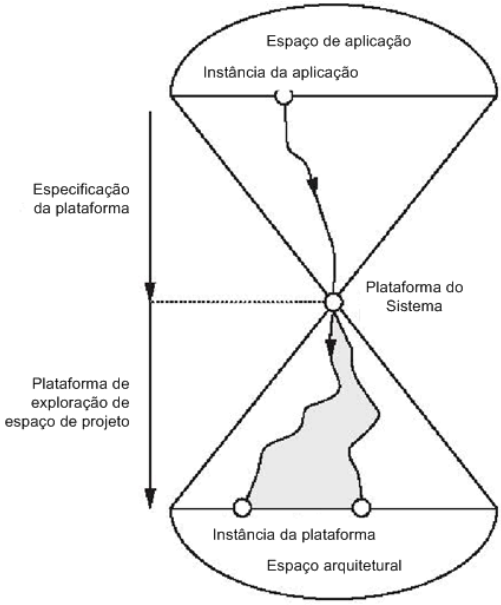


Figura 4.2: *Plataformas e Fluxo de Projeto na Metodologia PbD. O fluxo de projeto inclui a especificação da plataforma e exploração do espaço de projeto da plataforma. A plataforma do sistema representa a API dos desenvolvedores, que deve ser mapeada para a plataforma de sistema, que representa um conjunto de possíveis micro-arquiteturas (plataforma de hardware) que podem ser usadas para implementá-la.*
Fonte: [Sangiovanni-Vincentelli e Martin 2001]

Conforme é apresentado na figura 4.2, PbD trata três tipos distintos de plataforma: (1) Plataforma de hardware; (2) Plataforma de software; e (3) Plataforma de sistema. Nessa figura também pode-se observar, simplificadamente, o fluxo principal de projeto da PbD: Especificação da plataforma e Exploração do espaço de projeto da plataforma. A plataforma do sistema representa a API dos desenvolvedores, ou seja, a abstração para a plataforma de hardware. Assim, o desenvolvedor de uma (instância de) aplicação deve mapear essa aplicação para a plataforma de sistema (na etapa do fluxo de projeto denominada "especificação de plataforma"), que representa um conjunto de possíveis micro-arquiteturas (plataforma de hardware) que podem ser usadas para implementá-la. Quanto mais abstrata a plataforma de sistema, maior é o conjunto possível de instâncias de plataformas de hardware, mas torna-se mais difícil escolher a instância ótima e fazer automaticamente o mapeamento para ela. A escolha da plataforma de hardware final é feita na etapa de "exploração de espaço de projeto de plataforma", e pode ser feita para otimizar custo, eficiência, energia e flexibilidade [Keutzer et al. 2000]. Desse modo, a plataforma de sistema representa uma interface entre software e hardware, consistindo basicamente de um RTOS (RTOS - *Real-Time Operating System*), *device-drivers*, e sub-sistemas de comunicação, numa combinação das estratégias *top-down* e *bottom-up*, de modo que PbD é considerada um processo *meeting-in-the-middle*. Para a especificação da plataforma de sistema, PbD enfatiza a ortogonalização de conceitos de projeto, ou seja, a separação de função e arquitetura (separação vertical), e de comunicação e computação (separação horizontal).

PbD tem sido muito difundida e utilizada também para o projeto em nível de sistema (SLD - *System-Level Design*) [Balarin et al. 2003, Sangiovanni-Vincentelli 2007, Bergamaschi et al. 2008]. Ela baseia-se no uso de uma mesma plataforma arquitetural para muitas aplicações diferentes, ao invés de desenvolver uma nova plataforma para cada aplicação, e também no progressivo incremento de complexidade e detalhamento dos modelos em cada etapa, permitindo avaliar diferentes arquiteturas sem o uso de primitivas de baixo nível. Ainda conforme Keutzer et alli [Keutzer et al. 2000], modelos em nível de sistema capturam o comportamento da aplicação e algumas relações com a arquitetura, como escalonamento e alocação de recursos. Essa é uma característica-chave que permite que SLD seja aplicada nos estágios iniciais do projeto para realizar, por exemplo, exploração do espaço de projeto. Várias ferramentas que suportam SLD foram anunciadas na última década, incluindo Cadence VCC (VCC - *Virtual Component Co-design*) [Schirrmeister e Sangiovanni-Vincentelli 2001],

e Synopsys Co-Centric System Studio [Inc 2010].

É indiscutível o sucesso e a aplicabilidade da PbD. Contudo, ela levanta algumas questões que precisam ser consideradas. Essa metodologia baseia-em plataformas pré-concebidas e que podem ser reusadas em diferentes projetos (como placas de prototipação), mas não indica como as plataformas pré-concebidas são concebidas pela primeira vez. Também não indica qual grau de recursos desnecessários é aceitável para prover o reuso entre aplicações. Vale ressaltar que conceber reuso baseado em recursos desnecessários por possivelmente serem necessários a aplicações futuras aproxima-se da filosofia de computação genérica, e não da computação embarcada ou dedicada. A metodologia também não fornece indícios muito claros de como componentes de software podem ser projetados para a plataforma de sistema, e que características deve ter a API da plataforma de sistema de modo que muitas aplicações diferentes possam ser mapeadas a ela.

4.1.3 Projeto Dirigido pela Aplicação

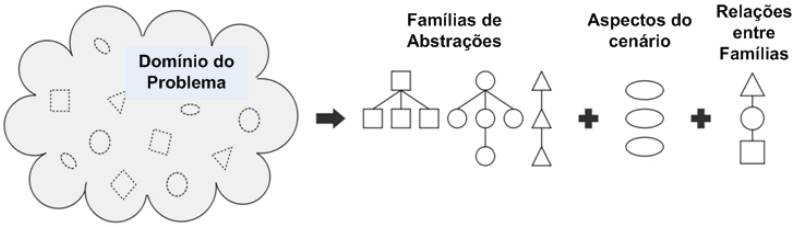
Sistemas embarcados dirigidos pela aplicação são compostos única e exclusivamente pelos componentes de software e de hardware necessários pela aplicação-alvo. A metodologia ADESD propõe estratégias para definir componentes que representem entidades significativas em diferentes sistemas e cenários de execução, facilitando seu reuso, o que inicia com a Engenharia do Domínio [Pietro-Diaz 1990]. As variações nas aplicações de certo domínio são tratadas como definido no Projeto Baseado em Famílias [Parnas 1976], ou seja, organizadas como membros de famílias de componentes. Entidades significativas do domínio que são independentes do cenário de execução podem ser modeladas pela ADESD como *abstrações do sistema*. Mesmo com uma separação criteriosa, essas abstrações ainda podem conter dependências relacionadas a requisitos não-funcionais e ao ambiente de execução ao qual são aplicadas (dependências arquiteturais). Assim, para reduzir essas dependências e aumentar a reusabilidade dos componentes, a ADESD utiliza três abordagens principais: (1) A *separação de aspectos*, conforme definido na Programação Orientada a Aspectos (AOP - *Aspect Oriented Programming*), para o isolamento dos requisitos não-funcionais; (2) *Adaptadores de cenário* [Fröhlich e Schröder-Preikschat 2000], que configuram e adaptam os componentes conforme o ambiente de execução, possivelmente aplicando *aspectos* ou

ativando *características configuráveis* [Tondello e Fröhlich 2005]; e (3) *Mediadores de hardware* [Polpeta e Fröhlich 2004], que encapsulam as dependências arquiteturais, mas diferem das HALs (HAL - *Hardware Abstraction Layer*) tradicionais por serem 'componentizados'.

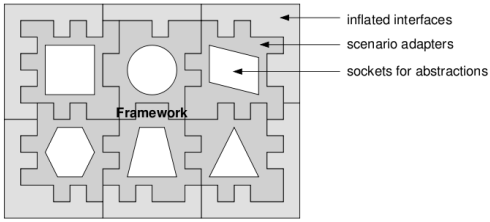
Essas estratégias e abordagens são representadas nas figuras 4.3(a) e 4.3(b). A figura 4.3(a) mostra a separação de entidades do domínio em famílias de abstrações. Os membros das famílias são os componentes. A configuração e adaptação dos componentes é mostrada na figura 4.3(b). Componentes possuem configurações próprias (*traits*), que correspondem, por exemplo, à frequência de uma CPU ou à taxa de transmissão de uma UART. Dependendo do cenário de execução, o próprio comportamento do componente precisa ser adaptado, normalmente devido a requisitos não-funcionais, como sincronismo e gerenciamento de energia, que exigem alterações em todos os componentes. Nesse caso, aspectos são aplicados aos componentes pelos adaptadores de cenário, conforme ilustrado ainda na figura 4.3(b). Também é possível que adaptações comportamentais sejam necessárias em apenas alguns componentes, caso em que outro tipo de aspectos, chamados na ADESD de *características configuráveis*, sejam aplicados apenas a componentes específicos. Conforme foi citado, isso isola as dependências arquiteturais e aspectos não-funcionais, o que evita a explosão na quantidade de componentes e facilita seu reuso.

A exemplo do OOD, ADESD sugere a classe (objeto) como a unidade básica de reuso, ou seja, dos componentes. Como exemplo da granularidade e composição de componentes da ADESD, considere uma possível aplicação embarcada muito simples que apenas execute uma região crítica de forma atômica, utilizando para isso um semáforo, conforme mostra o algoritmo 4.1. No trecho apresentado, a aplicação é composta unicamente pela *thread* `main`, que faz referência a métodos de uma *interface inflada*[1], `void P()` e `void V()`, e que compõe o único requisito da aplicação em relação ao sistema de suporte. Possíveis implementações de sistemas embarcados (software de suporte e hardware) que suportam essa aplicação podem ser parcialmente representados como um grafo de dependências entre componentes, conforme apresentado na figura 4.4. Conforme essa figura, os requisitos da aplicação (implementações para os métodos `void P()` e `void V()`) podem ser satisfeitos por um de dois componentes do repositório: `Semaphore_sw` ou `Semaphore_hw`, mas não am-

[1] uma *interface inflada* corresponde à união das interfaces de todos os componentes membros de uma família, como proposto por [Fröhlich 2001].



(a) Separação de entidades do domínio em famílias de componentes



(b) Configuração e adaptação de componentes. Adaptadores de cenário interceptam todas as invocações às interfaces dos componentes.

Figura 4.3: *Separação de entidades em famílias e configuração de componentes na ADESD. As abordagens propostas pela engenharia de domínio, Projeto Baseado em Famílias e Programação Orientada a Aspectos ajuda no isolamento de dependências arquiteturais e reuso dos componentes.*
Fonte: [Fröhlich 2001]

bos, formando uma restrição binária (equação (2.3) da seção 2.1). A dependência alternativa de um componente ou de outro é representada pela aresta tracejada, enquanto a aresta contínua representa uma dependência estrita. Cada um desses componentes, por sua vez, possui dependências de outros componentes do repositório, que devem ser sucessivamente satisfeitas[2]. O conjunto das escolhas entre dependências alternativas corresponde a parte de exploração do espaço de projeto na ADESD. Componentes que representam aspectos (que poderiam ser aplicados aos componentes apresentados) e aqueles relacionados à inicialização (`setup`, `memory map`, etc), não foram apresentados para facilitar a visualiza-

[2]ainda no exemplo da figura 4.4, por exemplo, o componente `Semaphore_sw` depende de `Synchronizer`, que depende de `Thread` e `CPU`, e assim sucessivamente.

ção da figura. Do exemplo apresentado conclui-se que representar o software embarcado, por mais simples que seja, com granularidade de processos ou tarefas/*threads* pode mascarar sua real complexidade.

```
1  int main(void) {
2     Semaphore sem = new Semaphore(1);
3     sem.P();
4     // this is a critical region
5     sem.V();
6     return 0;
7  }
```

Algoritmo 4.1: *Exemplo de uma aplicação embarcada simples, que apenas executa uma região crítica atomicamente, de forma que seus únicos requisitos sejam os serviços P() e V()*

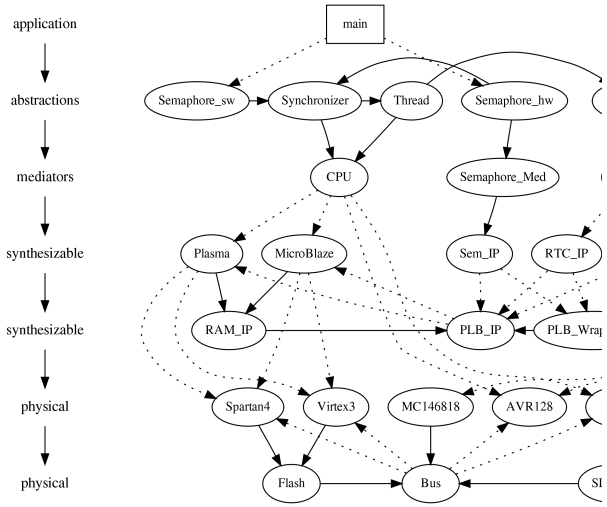


Figura 4.4: *Exemplo de dependência de componentes na ADESD. As linhas contínuas representam dependências normais entre componentes, enquanto as linhas pontilhadas representam dependências mutuamente exclusivas. Os componentes apresentados ilustram a granularidade das entidades reusáveis.*

A metodologia ADESD guia o projetista desde a criação de componentes reusáveis que irão compor um repositório (infra-estrutura) até a geração do sistema final a partir de uma aplicação embarcada. Ela permite criar componentes

mais reusáveis e eficientes por iniciar com a engenharia de domínio. Isolando características não-funcionais em aspectos, reduz o número de componentes e os torna mais adaptáveis e reusados em diferentes cenários de execução, o que isola o fluxo da infra-estrutura do fluxo da aplicação, e permite ao desenvolvedor da aplicação concentrar-se unicamente na aplicação, e não no suporte arquitetural [Marcondes et al. 2006]. Com isso, ADESD também não especifica um fluxo de projeto de aplicação, mas sugere a utilização dos mesmos fluxos do desenvolvimento utilizados pela moderna engenharia de software: OOD e MDA. A utilização de metaprogramação estática[3] para configuração dos componentes em tempo de compilação diminui ou mesmo elimina o *overhead* do desenvolvimento orientado a objetos [Fröhlich 2001]. Representando componentes de software e de hardware, permite a exploração de novos suportes de hardware. Aqui, cabem ainda duas colocações importantes: (1) Até o desenvolvimento desta tese, apenas componentes de software e de hardware sintetizável (*IP cores*) foram incorporados usando ADESD. Esta tese propõe a incorporação também de componentes físicos; e (2) Até o desenvolvimento desta tese não havia suporte computacional para a exploração do espaço de projeto na suite de ferramentas da ADESD. O modelo que permite tal exploração é o alvo desta tese e foi incorporado à suite de ferramentas de suporte ao projeto de sistemas embarcados com a ADESD.

EPOS e Suite de Ferramentas de Suporte à ADESD

Um dos primeiros produtos desenvolvido pelo uso da ADESD foi o EPOS (EPOS - *Embedded and Parallel Operating System*), um *framework* que pode ser usado em diferentes ambientes[4], provendo transparência arquitetural ao suporte a aplicações embarcadas [Fröhlich 2001]. Como produto da ADESD, o EPOS é baseado em componentes adaptáveis e reusáveis, que representam entidades significativas no domínio dos sistemas operacionais embarcados. Além de componentes do sistema operacional, o EPOS foi estendido para tratar também com hardware sintetizável [Polpeta e Fröhlich 2004], permitindo o uso de componentes cuja implementação possa ser mapeada em software ou hardware sintetizável.

A utilização da ADESD no projeto de sistemas operacionais e o suporte de hardware apresenta a vantagem de permitir configuração e geração automáticas.

[3] *metaprogramação* estática, ou programação genérica, é um paradigma de programação no qual os algoritmos são escritos em uma gramática estendida de forma a adaptar-se através da especificação das partes variáveis que são definidas na instância do algoritmo.

[4] Atualmente, EPOS permite gerar instâncias de sistemas embarcados para diversas plataformas diferentes, incluindo IA32, PPC405, SparcV8 (Leon2), MIPS (Plasma), AVR8 e ARM.

O conceito de interface inflada (do projeto Baseado em Famílias) utilizado pela ADESD permite que o suporte computacional (sistema operacional + suporte de hardware) seja gerado automaticamente a partir de um conjunto de componentes de hardware e software, pois as interfaces infladas servem como especificação de requisitos para o sistema a ser gerado. Essa abordagem baseia-se na configuração estática (em tempo de compilação), permitindo a geração de versões otimizadas e específicas do suporte computacional para cada aplicação, em contraposição ao projeto baseado em plataforma. Devido à sua complexidade, como ocorre em outras metodologias, a geração de sistemas embarcados precisa de ferramentas computacionais de suporte.

A estrutura geral da suite de ferramentas de suporte à ADESD é apresentada na figura 4.5, e mostra que ela é composta por 3 (três) ferramentas principais: `Analisador`, `Configurador` e `Gerador`. A especificação da aplicação deve ser escrita com base nas interfaces dos componentes de um repositório, que formam a API do EPOS. Essa especificação é submetida inicialmente ao `Analisador`. Essa ferramenta procura por referências a essas interfaces[5], e então cria uma especificação de requisitos composta por métodos, tipos e constantes usados pela aplicação. Se um requisito pode ser satisfeito por um único componente do repositório, o `Analisador` pode selecioná-lo automaticamente. Se mais de um componente satisfaz o requisito, é necessário que o projetista selecione, manualmente, um deles.

Os componentes escolhidos com a ajuda do `Analisador` são adicionados ao `Configurador` e usados para construir uma árvore de dependências do sistema, que representa uma alternativa de projeto. Arquivos com regras de composição e dependências, e adaptadores de cenários também são usados para esse fim. O `Configurador` então pode manter informações sobre dependências da aplicação em relação aos componentes do sistema, sobre dependências dos componentes entre si e sobre suas características, e sobre as regras que devem ser seguidas para criar a configuração. A estratégia usada para descrever componentes no repositório e suas dependências tem papel fundamental para tornar o processo de configuração possível. A descrição dos componentes deve ser completa o suficiente para que o `Configurador` possa identificar automaticamente quais abstrações satisfazem melhor os requisitos da aplicação sem gerar conflitos ou configurações e composições inválidas. A descrição das inter-

[5] O `Analisador` aplica uma técnica que envolve a compilação do código-fonte da aplicação, a análise dos arquivos-objeto resultantes, e a identificação de símbolos não resolvidos relacionados a métodos e constantes do `namespace 'System'`.

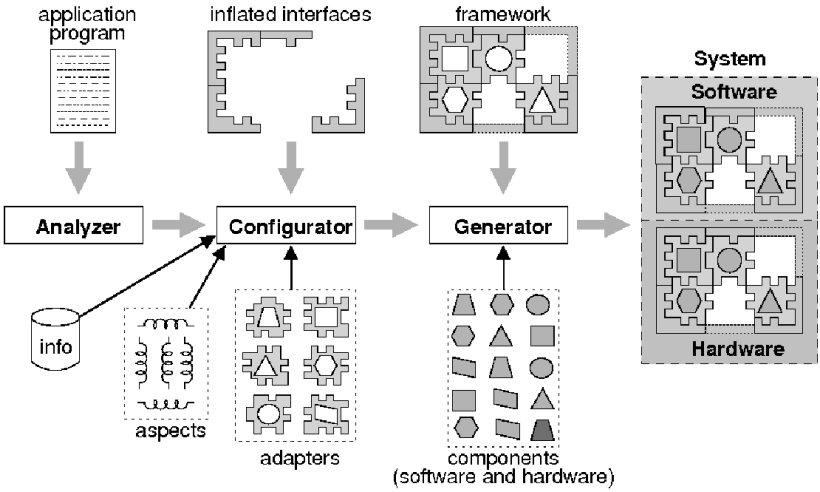


Figura 4.5: *Processo de geração de sistemas embarcados pela ADESD. O Analisador extrai as interfaces infladas da aplicação, enquanto o Configurador escolhe os componentes que satisfazem tais interfaces, e os configura. Por fim, o Gerador utiliza ferramentas de terceiros para compilar o software e sintetizar o hardware, gerando o sistema embarcado final.*
Fonte: [Fröhlich 2001]

faces numa família e de seus membros é a principal fonte de informação para o `Configurador`, mas a montagem de um sistema baseado em componentes vai muito além da verificação sintática de conformidade com essas interfaces. Propriedades não funcionais e comportamentais também precisam ser cobertas. Para isso, a descrição do componente inclui dois elementos especiais: `feature` e `dependency`. Esses elementos podem ser aplicados para especificar características providas pelos componentes e dependências entre componentes que não podem ser diretamente deduzidas a partir de suas interfaces. A descrição dos componentes inclui também uma informação abstrata de custo (`cost`), que pode ser usada para a seleção de componentes alternativos. A interpretação do que significa o "custo" de um componente é deixada a cargo do projetista. Mais detalhes e exemplos da descrição dos componentes podem ser encontrados em [Tondello e Fröhlich 2004, Tondello e Fröhlich 2005].

O último passo no processo de desenvolvimento do sistema é realizado

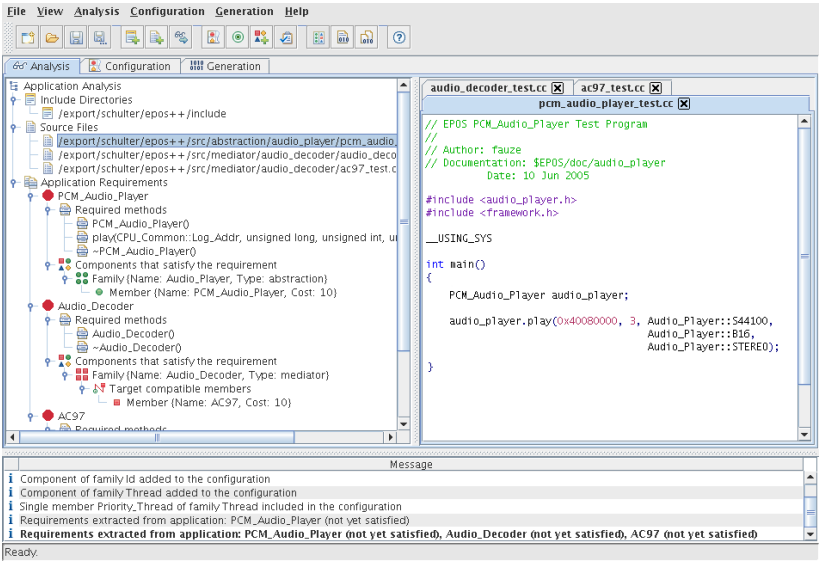


Figura 4.6: *Tela do Analisador. Mostra a aplicação-alvo (à direita) e a árvore com interfaces infladas extraídas a partir dela (à esquerda), incluindo os componentes que podem executar os serviços solicitados pelas interfaces.*
Fonte: [Schulter et al. 2007]

pelo `Gerador`. Internamente, ele gera um conjunto de chaves que associam cada interface invocada pela aplicação a um componente específico existente no repositório, e ativa os aspectos de cenário (da AOP) que foram eventualmente identificados pelo `Configurador` como necessários. Em relação ao hardware, o `Gerador` produz uma lista de mediadores que foram incluídos pelo `Configurador`, especificando quais estão associados a IPs. O `Gerador` então traduz as chaves em parâmetros para um *framework* de componentes metaprogramados e executa a compilação do sistema. Além disso, quando o suporte de hardware é configurável (ex. FPGA), o `Gerador` produz o arquivo de configuração da síntese que guarda os parâmetros para os IPs e as informações necessárias para interconectar os IPs ao SoC. Esse arquivo de configuração é escrito numa linguagem de descrição de hardware e, juntamente com os IPs selecionados, são passados a uma ferramenta de síntese de terceiros, que realiza a tradução da especificação num arquivo *bitstream* de configuração da FPGA-alvo.

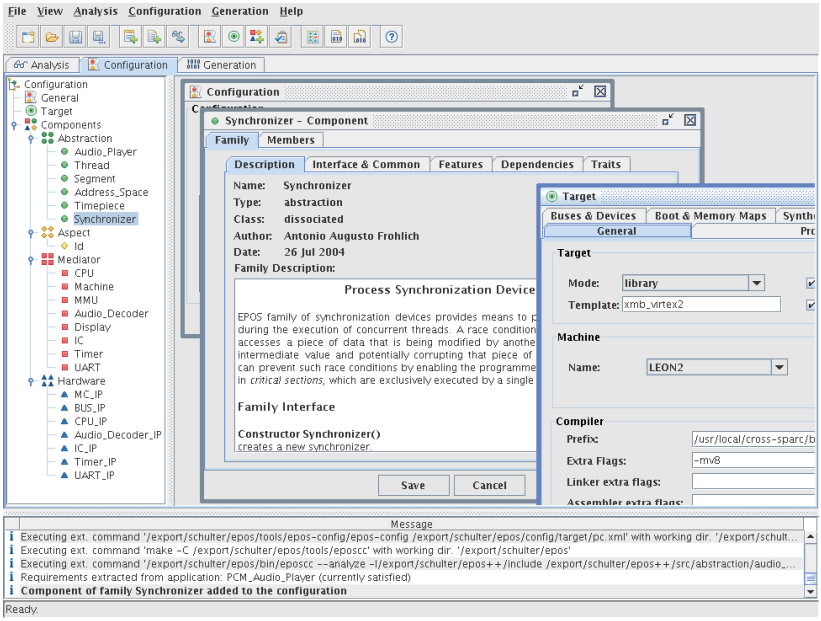


Figura 4.7: *Tela do Configurador. Na parte esquerda são visíveis a lista dos componentes utilizados no projeto do sistema embarcado e que implementam as interfaces infladas, separados por seu "tipo" (abstraction, aspect, mediator, hardware). À direita são visíveis as várias características de alguns desses componentes, e permite que o projetista, manualmente, selecione os componentes e os configure.*
Fonte: [Schulter et al. 2007]

O processo descrito acima corresponde à versão imediatamente anterior ao desenvolvimento desta tese, cujas contribuições principais correspondem exatamente ao desenvolvimento de modelos e metamodelos para a exploração do espaço de projeto de sistemas embarcados dirigidos pela aplicação, e que foram integrados à nova suite de ferramentas da ADESD e mapeados a um modelo evolucionário com adaptações que é utilizado para um processo de otimização multiobjetivo, usado nessa exploração. Desse modo, entre os avanços desenvolvidos no processo descrito, destacam-se os seguintes: (1) O metamodelo de componentes da ADESD passa a suportar, além de componentes lógicos/*cyber* (software e hardware sintetizável), também componentes físicos (eletro-eletrônicos

e mecânicos), encapsulamentos físicos dos componentes, ferramentas de software (compiladores, sintetizadores, depuradores, simuladores, e outras) e suportes de hardware. A estratégia para descrever tais componentes (essencial para o `Configurador`), foi remodelada e expandida; (2) Componentes não precisam ser descritos exclusivamente em C++ ou VHDL. Com a representação de ferramentas de software, eles podem ser especificados em outras linguagens e níveis de abstração, incluindo FSM (FSM - *Finite State Machine*) ou SystemC; (3) Diversas escolhas feitas manualmente pelo projetista, como a plataforma-alvo e escolha final dos componentes, passam a ser automatizadas pela etapa de exploração de espaço de projeto; e (4) Características descritas nos componentes podem ser utilizadas para compor funções-objetivo a serem minimizadas no processo de exploração.

4.2 Exploração do Espaço de Projeto

Nesta seção são apresentados alguns aspectos gerais da exploração em sistemas embarcados e de suas etapas. Também são apresentadas as principais estratégias utilizadas nos trabalhos relacionados encontrados na literatura, destacando o escopo dessas explorações, suas limitações e pontos relevantes a esta tese.

4.2.1 Aspectos Gerais

A exploração do espaço de projeto (DSE - *Design Space Exploration*) é um aspecto central no projeto de sistemas embarcados. Tipicamente, DSE é um problema de otimização multiobjetivo (MOP - *Multi-Objective Problem*) onde consumo de energia, desempenho, área de silício, tamanho de memória e outros critérios são os objetivos a ser otimizados para encontrar soluções ótimas de Pareto [Chakraborty, Mitra e Roychoudhury 2006]. A quase totalidade das pesquisas, todavia, otimiza apenas dois ou três objetivos simultaneamente, como área de silício e desempenho [So, Diniz e Hall 2003, Beltrame et al. 2006, Pilato et al. 2008], ou tempo de execução e consumo de energia [Mohanty et al. 2002, Palesi e Givargis 2002], dadas as dificuldades em otimizar simultaneamente mais objetivos ou mesmo obter estimativas para outras características do sistema que possam ser otimizadas.

Um sistema embarcado é constituído de software embarcado e de elementos arquiteturais, que compõem o suporte de hardware sobre a qual executa o software embarcado. Existem diversas possíveis implementações para um sistema embarcado, no qual os elementos do software embarcado (processos, *threads*, métodos, blocos básicos ou mesmo instruções de máquina) são mapeados para elementos da arquitetura (nodos, microprocessadores, dispositivos de lógica programável, memórias, barramentos, somadores, multiplicadores). Assim, de forma consensual, a DSE é tratada como consistindo de três etapas [Micheli 1994, So, Diniz e Hall 2003, Haubelt e Teich 2003, Schlichter et al. 2006, Pilato et al. 2008, Ferrandi et al. 2008]:

- o *mapeamento*, que associa elementos de software a elementos de hardware;
- a *alocação*, que indica quais elementos de hardware são usados na implementação; e
- o *escalonamento*, que associa um tempo de início de execução a cada elemento de software.

O processo de exploração do espaço de projeto é dificultado pois as entidades que são representadas como elementos de software ou elementos da arquitetura nessas três etapas podem ser bem distintas, dado que a DSE pode ser realizada em diferentes níveis de abstração do projeto, que compreendem principalmente os seguintes [Ku, Thiele e Zitzler 2005]:

- *sistema*, em que é escolhida a composição completa do sistema por componentes, o mapeamento das aplicações (elementos de software) a nodos do sistema (elementos de hardware) e a infra-estrutura global de comunicação.
- *tarefas*, em que as aplicações são particionadas em processos ou *threads* e são escolhidas técnicas de escalonamento, gerenciamento de memória e arbitragem, entre outras.
- *compilação*, em que são escolhidas técnicas de compilação que podem beneficiar tempo de execução, espaço de memória ou consumo de energia. Entre algumas técnicas usadas estão *loop unrolling* e *function inlining*, que operam sobre elementos de software como procedimentos, laços ou blocos

básicos, e consideram elementos de hardware como o conjunto de instruções do processador, tamanho de memória e de cache.

- *arquitetura*, em que são escolhidas as unidades de processamento e suas unidades funcionais (pipeline, execução em ponto-flutuante, etc), tamanho de memória principal e cache e estrutura de comunicação do nodo.

- *projeto lógico*, em que são escolhidos largura de barramentos e de dados e outras decisões que podem beneficiar área de silício, latência ou consumo de energia. Os elementos alocáveis são registradores e operadores, mapeados em unidades funcionais da arquitetura, como somadores e multiplicadores, e o escalonamento associa o início de cada operação a um pulso de clock [Micheli 1994].

A quantidade de alternativas a serem exploradas em cada nível impossibilita que esse problema seja tratado de uma única vez, e demanda uma abordagem hierárquica [Mohanty et al. 2002] ou a exploração de apenas parte do espaço de projeto, normalmente limitado a um desses níveis de abstração. Infelizmente, as escolhas realizadas em cada nível de abstração dependem e influenciam também de escolhas nos demais níveis, o que torna este um problema no qual pode ser necessário voltar a estágios anteriores após escolhas feitas em estágios mais avançados, num processo hierárquico iterativo. Uma forma de reduzir a quantidade de iterações é diminuir a interdependência de decisões em cada nível [Han et al. 2007]. Diversas pesquisas têm realizado a DSE de forma hierárquica, mas poucas têm se preocupado em como reduzir as interdependências entre os níveis [Han et al. 2007]. Felizmente, também pode-se dizer que as técnicas usadas para DSE num nível de abstração são aplicáveis aos demais níveis, de forma que essas técnicas são "reusáveis" e podem ser classificadas de forma ortogonal ao nível de abstração do projeto ao qual são aplicadas. Na opinião deste autor, a exploração hierárquica pode ser auxiliada se um único modelo for usado para representar as decisões nos diferentes níveis de abstração, mesmo que as decisões não sejam tomadas todas de uma vez, dada a grande quantidade de alternativas. Partes diferentes do modelo poderiam ser fixadas ou então permitidas variar, dependendo do nível de abstração sendo explorado, o que é feito nesta tese.

4.2.2 Estratégias de Exploração do Espaço de Projeto

A DSE consiste de dois sub-problemas: modelar o sistema e seu espaço de projeto, e então procurar nesse espaço de projeto pelas melhores soluções que atendem os objetivos de projeto [Azizi et al. 2010]. Tal modelo do sistema embarcado no nível (ou níveis) em que se deseja explorar, pode incluir software, hardware, alocação e mapeamento entre software e hardware, circuitos digitais, e possivelmente outros elementos e até aspectos mecânicos do sistema. Tal modelo deve servir de base a alguma estratégia que avalie diferentes alternativas e combinações nesses elementos e escolha a melhor solução, respeitando possíveis restrições impostas. As estratégias para tal avaliação correspondem a técnicas de otimização. O modelo que representa o sistema embarcado e a técnica de otimização utilizada estão intimamente relacionados, sendo normalmente o primeiro definido em termos do segundo. Técnicas de otimização podem ser determinísticas ou estocásticas, e podem visar a otimização de uma ou várias funções-objetivo simultaneamente. O capítulo 2 tratou especificamente de otimização multiobjetivo, de maior interesse para o projeto de sistemas embarcados, e a seção 2.4 apresentou as principais técnicas de otimização multiobjetivo.

Explorar todo o espaço de projeto costuma ser inviável, devido à enorme quantidade de soluções (configurações) possíveis. Como um exemplo simples, suponha a aplicação embarcada apresentada na seção 4.1.3, figura 4.4, que possui uma *thread* que apenas faz referência aos métodos de um semáforo (`void P(), void V()`). Se cada um dos 25 componentes envolvidos tiver uma única configuração com apenas 2 valores possíveis, mais as 11 dependências alternativas (com 2, 3 ou 4 alternativas cada), então a quantidade de soluções possíveis será de $2^5.3^1.4^2.2^{25} = 5,154.10^{10}$ soluções. Se a avaliação de cada solução consumir 1 segundo, seriam necessários mais de mil e seiscentos anos para avaliar todas as soluções possíveis. Assim, estratégias de DSE devem manter aceitável o tempo de exploração e ainda assim garantir boa qualidade das soluções, o que pode ser conseguido com duas abordagens diferentes: diminuir a quantidade de soluções avaliadas e/ou diminuir o tempo de avaliação de cada solução [Ascia et al. 2006, Calborean e Vintan 2010]. O emprego dessas abordagens é feito, então, ou no algoritmo utilizado para seleção e otimização das soluções avaliadas, ou nas estratégias para avaliação das funções-objetivo dessas soluções, ou em ambos [Ascia et al. 2007]. De modo geral, as estratégias utilizadas incluem fixar alguns elementos do sistema para diminuir a quantidade de explorações, como realizar a exploração num único nível de abstração ou então fazer

uma exploração hierárquica utilizando métodos mais rápidos (e menos precisos) nos níveis mais altos, diminuindo o tempo de avaliação médio de cada avaliação.

4.2.3 Exploração em um Nível

Uma forma de diminuir a quantidade de soluções possíveis é realizar a exploração apenas num nível de abstração, fixando os elementos dos demais níveis. Um caso comum é fixar a arquitetura-alvo, como sugerido pela metodologia de projeto baseado em plataforma (PbD - *Platform-Based Design*), em que a plataforma é pré-definida, mas parametrizável (frequência de CPU, tamanho de memória, etc) [Sangiovanni-Vincentelli 2005], permitindo seu reuso em diferentes projetos. Essa abordagem reduz a complexidade da DSE nos níveis hierárquicos de arquitetura e projeto lógico, que limita-se a parametrizar a plataforma para minimizar alguns objetivos e algumas vezes também mapear os processos quando o hardware possui mais de um processador. É nos níveis mais altos que a exploração baseada em plataforma costuma focar. Várias pesquisas encontradas na literatura tem seguido essa abordagem para diminuir a complexidade da DSE. Umas poucas utilizam técnicas clássicas e determinísticas para a otimização envolvida na exploração, enquanto a maioria já faz uso de técnicas modernas e estocásticas.

Entre as pesquisas que fazem a exploração usando técnicas clássicas, incluem-se a de Palermo et alli [Palermo, Silvano e Zaccaria 2008], que utilizam projeto de experimentos (DoE - *Design of Experiments*) e superfície de resposta (RSM - *Response Surface Modeling*) para a DSE em arquiteturas do tipo multiprocessadores-em-chip. Inicialmente, o DoE é usado para selecionar um conjunto de configurações correspondentes aos experimentos a serem feitos, e que fornecem uma visão macroscópica da superfície do espaço de projeto. Os experimentos são realizados, ou seja, os vetores de decisão selecionados são avaliados, e a fronteira de Pareto é obtida para a superfície do espaço atual. Então, técnicas de RSM são usadas para gerar uma nova superfície de projeto, através de novas funções de regressão ou interpolação estimadas com base nos experimentos realizados, e obtém-se uma nova fronteira de Pareto. O processo é repetido iterativamente até que um critério de parada seja atingido. Para obter a fronteira de Pareto tratando soluções inviáveis, criam um novo operador de *dominância restrita* $x_i \lhd x_j$, definido formalmente na equação (4.1)

$$x_i \lhd x_j = \begin{cases} x_i \prec x_j & \text{, se } \textit{VIAVEL}(x_i) \\ \text{falso} & \text{, se } \neg\textit{VIAVEL}(x_i) \wedge \textit{VIAVEL}(x_j) \\ x_i \ll x_j & \text{, se } \neg\textit{VIAVEL}(x_i) \wedge \neg\textit{VIAVEL}(x_j) \end{cases} \quad (4.1)$$

onde *VIAVEL* é uma função booleana que indica se uma solução violou alguma restrição de projeto ou não, e o operador '$\ll$' é usado para comparar duas soluções inviáveis, e combina a ordenação, penalidade e dominância de Pareto das duas soluções para formar um único valor de dominância.

Entre as diversas pesquisas que utilizam técnicas estocásticas para a exploração, incluem-se Noonan e Flanagan [Noonan e Flanagan 2006], que utilizaram uma arquitetura baseada no processador Intel IXP1200 para fazer a DSE em nível de arquitetura, configurando parâmetros como frequência (IX, MAC, SDRAM) e largura dos barramentos. Eles utilizaram um algoritmo genético (GA - *Genetic Algorithm*) para minimizar a área de silício e o consumo de energia, sendo que as funções-objetivo para esses custos foram obtidas empiricamente e o operador de mutação age de forma diferenciada em genes que representam largura de barramento ou frequência de operação. Em ambos os casos o operador pode somar ou subtrair um valor que especificado pelo projetista. Três estratégias de evolução foram comparadas: a de Darwin, de Lamark e de Baldwin.

Nessa pesquisa, um operador de mutação específico foi definido para tratar das variáveis que representam o espaço de projeto. Embora tenham utilizado uma técnica interessante, permitindo o projetista especificar o valor somado a um valor aleatório gerado pela mutação, o que permite controlar a "distância", ou o efeito da mutação, este autor não considera que seja de responsabilidade do projetista definir esse valor. O efeito da mutação sobre diferentes genes deveria ser ajustado automaticamente pela própria técnica de evolução. A estimação da qualidade foi feita pelas funções empíricas criadas, que são muito mais rápidas que co-simulação, diminuindo o tempo de avaliação de cada solução, mas possivelmente não representam bem as funções-objetivo reais. Técnicas matemáticas, como as utilizadas na pesquisa apresentada a seguir, tem sido mais comuns e são mais adequadas.

Anderson e Khalid [Anderson e Khalid 2006] utilizaram uma plataforma fixa baseada no *softcore* Nios da Altera e a exploração é usada para definir os parâmetros da arquitetura e minimizar a área e a latência do circuito, fazendo exploração em nível de projeto lógico. Para estimar os custos de área e latência, inicialmente geraram 47 configurações diferentes e sintetizaram os circuitos, depois

usaram os resultados da síntese para criar uma equação de regressão parabólica que passou a ser usada por um algoritmo genético (GA) durante a DSE.

Haubelt et alli [Haubelt et al. 2008] fazem a DSE em nível arquitetural e de projeto lógico sobre uma arquitetura baseada unicamente em FPGA. A especificação de uma aplicação é feita em SystemC, e é limitada ao domínio de aplicações multimídia, devendo ser escrita usando a biblioteca SysteMOC [Falk et al. 2006]. A aplicação é transformada num modelo SysteMOC, que pode então ser transformado em software (por simples manipulações no código) ou em hardware (usando Forte Cinthesizer [Systems 2009]). Um *template* de arquitetura especifica todos os processadores, memórias, dispositivos, barramentos e conexões possíveis numa arquitetura baseada unicamente em FPGA, e a DSE deve escolher uma arquitetura ótima desse *template*. A DSE é feita com base na aplicação especificada em SysteMOC e também no hardware sintetizado pelo processo descrito, e busca soluções ótimas em termos de *throughtput*, latência e área de silício, utilizando algoritmos evolucionários para encontrar soluções de Pareto. Estimativas de área são extraídas da ferramenta de síntese e de *throughtput* e latência através do *framework* VCP [Streubühr et al. 2006]. A exploração de 7.600 soluções numa estação com 2,4GHz levou mais de 2 dias e meio para ser concluída (aproximadamente 30 segundos por avaliação).

Embora essa pesquisa faça a exploração nos níveis de arquitetura e de projeto lógico, não se trata de exploração hierárquica. Um aspecto interessante nessa pesquisa é o uso de um *template* de arquitetura. Embora representando apenas arquiteturas baseadas em FPGA na pesquisa apresentada, diferentes *templates* poderiam ser usados para exploração de suportes de hardware, ou para definir agrupamentos comuns entre componentes (incluindo componentes de software), como é feito nesta tese. Outro aspecto interessante da pesquisa apresentada é a modelagem da aplicação em SystemC, que pode ser transformada em software ou em hardware. Este autor considera que a representação de componentes híbridos[6] é indispensável ao projeto de sistemas embarcados, mas considera também que o software embarcado é subestimado em muitas pesquisas, principalmente aquelas que focam em nível de arquitetura e de projeto lógico.

Bauer et alli [Bauer, Shafique e Henkel 2009] desenvolveram uma ferramenta para a exploração do espaço de projeto em nível arquitetural, explorando processadores reconfiguráveis através da avaliação de diferentes parâmetros ar-

[6]um *componente híbrido* é um componente que podem ser mapeado tanto para software quanto para hardware.

quiteturais. Os autores informam ser a primeira ferramenta de exploração de processadores reconfiguráveis com granularidade fina com uma base de comparação confiável [Bauer, Shafique e Henkel 2009]. Entretanto, não é uma ferramenta automática de exploração do espaço de projeto, mas um simulador com um conjunto de ferramentas que permite a comparação do impacto causado pelos parâmetros dos processadores. Assim, nenhuma busca pela fronteira de Pareto ou qualquer otimização mono ou multiobjetivo é realizada. Parâmetros das diferentes soluções são alterados, as soluções são simuladas e então os resultados são tabulados e comparados. Este autor não considera que esse tipo de abordagem seja, efetivamente, aquilo que é denominado nesta tese de exploração do espaço de projeto; contudo, a pesquisa citada não é a única a afirmar que realiza essa etapa, mesmo sem qualquer busca por soluções ótimas.

Embora muitas outras pesquisas possam ser citadas, considero que as pesquisas previamente apresentadas representam bem a abordagem atual típica: plataformas pré-existentes e fixas, apesar de parametrizáveis, são utilizadas; A exploração pode se dar sobre os parâmetros dessa plataforma (caso mais comum) ou então sobre um nível mais alto, normalmente compilação ou sistema, em que o modelo de software é muito simplificado. Alguns resultados mais precisos, obtidos a partir de co-simuladores normalmente em nível de projeto lógico RTL (RTL - *Register Transler level*) são utilizados para obter uma amostragem que é usada para gerar modelos de estimação mais rápidos (e menos precisos), como equações de regressão ou ainda lógica difusa ou redes neurais [Oyamada et al. 2007]. O objetivo ainda é diminuir o tempo necessário para a avaliação de cada solução. Este autor considera que a diminuição do tempo de avaliação é imprescindível para que a exploração necessária ocorra em tempo aceitável, mas também considera que as abordagens utilizadas não capturam adequadamente o comportamento de muitos sistemas embarcados, incluindo aqueles em que aspectos dinâmicos da execução do software influenciam os objetivos a serem minimizados, o que parece ser comum. Uma solução adequada para esse problema extrapola o escopo desta tese.

4.2.4 Exploração Hierárquica

Também na exploração hierárquica encontram-se pesquisas realizando a otimização associada à exploração do espaço de projeto através de técnicas determinísticas e também através de técnicas estocásticas. Entre as pesquisas que uti-

lizam técnicas determinísticas de otimização, estão Han et alli [Han et al. 2007], que apresentam um método para DSE hierárquica que visa reduzir a quantidade de interdependências entre decisões tomadas em cada nível de abstração. Uma especificação é representada por um grafo hierárquico acíclico $\{\{HG_k^i(V_k,E_k)\}_{i=1,2,...,m}\}_k$, em que cada vértice V representa uma tarefa ou um sub-grafo e cada aresta E representa relações de dependência de dados ou comunicação entre as tarefas. O i-ésimo grafo do nível de abstração k é representado por $HG_k^i(V_k,E_k)$ e corresponde a um nodo-folha de um j-ésimo grafo no nível $k-1$. A otimização de um grafo i é dada por $O_k^i = f^i(I_k^i)$ sujeito a Cs_i^k , $i = 1,2,...,M_k$, onde O_k^i, I_k^i e Cs_i^k são os parâmetros de saída, de entrada e as restrições do grafo i no nível k, respectivamente. f^i é a função a ser otimizada nesse grafo e M_k é a quantidade de grafos no nível k. O modelo de otimização, então, consiste na especificação hierárquica, um conjunto de restrições e um conjunto de recursos como entrada, e produz um conjunto de Pareto no espaço de soluções em cada nível de abstração, que é usado como entrada no nível seguinte. Desta forma, os níveis menos abstratos têm, como vetores de entrada, apenas soluções otimizadas em níveis anteriores, ou seja, $O^{i-1} = f(O^i) = f(f(I^i))$. O método é apresentado e provado matematicamente, mas não há resultados práticos.

A ideia de refinamentos sucessivos entre níveis de abstração utilizando soluções do conjunto de Pareto em um nível como entrada para o nível seguinte é interessante, e diminui a quantidade de alternativas a serem exploradas nos níveis mais baixos, além de diminuir a interferência entre as decisões tomadas em níveis de abstração deferentes. Entretanto, considero que a representação do sistema usando grafos e a otimização determinística trazem limitações práticas à sua utilização. Este autor propõe a utilização de abordagem análoga, mas utilizando um modelo mais adaptável e que também deve permitir a posterior melhoria das soluções de Pareto em níveis mais altos, previamente determinadas, já que as decisões num nível afetam as decisões em outros níveis. Porém, modelar um sistema embarcado em vários níveis de abstrações usando grafos hierárquicos não é de fácil aplicação. Portanto, o modelo usado nesta tese foi um algoritmo evolucionário.

Entre as pesquisas que realizam a otimização através de métodos estocásticos, estão Schlichter et alli [Schlichter, Haubelt e Teich 2005, Schlichter et al. 2006], mesmo utilizando estruturas normalmente utilizadas em técnicas clássicas de otimização para modelar o problema. Eles especificam o problema de DSE como um grafo de especificação $G_s(V_s,E_s)$ que consiste de um grafo de processos $G_p(V_p,E_p)$ e um grafo de arquitetura $G_a(V_a,E_a)$ e um vetor de arestas de mapeamento E_m, tal que $V_s = V_p \bigcup V_a$, $E_s = E_p \bigcup E_a \bigcup E_m$ e

$E_m \subseteq V_p \times V_a$ representa "o processo v_p pode ser implementado pela arquitetura v_a". Um exemplo dessa especificação pode ser vista na figura 4.8(a). A ativação de uma especificação é uma função $a : V_s \times E_s \mapsto \{0,1\}$ que associa a cada aresta $e \in E_s$ e a cada vértice $v \in V_s$ o valor 1 (ativado) ou 0 (inativado). Uma alocação α é o subconjunto de todos os vértices e arestas ativados da arquitetura, formalmente apresentada na equação (4.2). Um mapeamento β é viável se e apenas se as três condições das equações (4.3), (4.4) e (4.5) forem satisfeitas:

$$\alpha = \{v \in V_a | a(v) = 1\} \bigcup \{e \in E_m | a(e) = 1\} \tag{4.2}$$

$$\forall e = (v_p, v_a) \in \beta : v_a \in \beta \tag{4.3}$$

$$|\{e \in \beta | e = (v_p, v_a), v_a \in V_a\}| = 1 \tag{4.4}$$

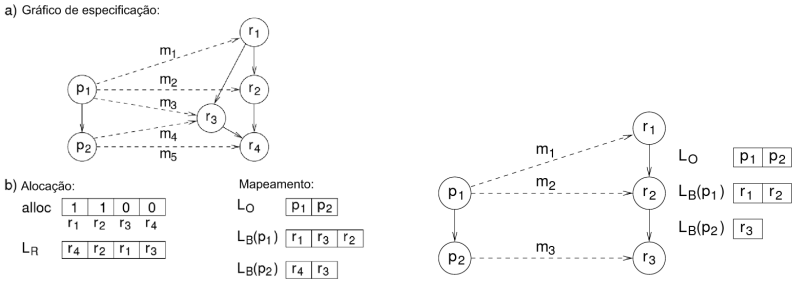
$$\forall e \in (v_i, v_j) \in E_p : \quad \begin{array}{l} (\exists (v_i, v), (v_j, v) \in \beta) \vee \\ (\exists (v_k, v_l) \in E_a \bigcap \alpha, (v_i, v_k), (v_j, v_l) \in \beta) \end{array} \tag{4.5}$$

Essas condições significam que cada aresta ativada do mapeamento termina num vértice ativado (eq. (4.3)), para cada processo há apenas uma aresta de mapeamento ativada (eq. (4.4)) [7] e, ou ambas as operações são mapeadas para o mesmo vértice v ou existe uma aresta ativada no grafo de arquitetura que representa um canal de comunicação entre os vértices onde as operações estão mapeadas (eq. (4.5)). Um MOEA foi usado para fazer a DSE sobre essa especificação, gerando uma implementação ψ viável, que é uma tripla $\psi = (\alpha, \beta, \gamma)$, onde α é uma alocação viável, β um mapeamento viável e γ é um escalonamento (figura 4.8(b)).

Essa pesquisa utiliza um modelo baseado em grafos para representação de um sistema embarcado. Esse modelo é mapeado num modelo evolucionário sobre o qual se dá a exploração do espaço de projeto, como proposto nesta tese. Porém, considero que o modelo de sistema embarcado apresentado é limitado, e foca basicamente na alocação e no mapeamento, e verificações de viabilidade e consistência das soluções são necessárias. Proponho o uso de um modelo mais abrangente de um sistema embarcado e uma abordagem com inspiração na GEP para garantia a priori da consistência da alocação e do mapeamento.

Outra pesquisa representativa da literatura é a de Mohanty e Davis [Mohanty et al. 2002], que propõem uma metodologia para exploração hierárquica do espaço de projeto, utilizando técnicas de avaliação mais rápidas e menos

[7] nessa equação, o operador $|\{\}|$ representa a quantidade de elementos no conjunto.

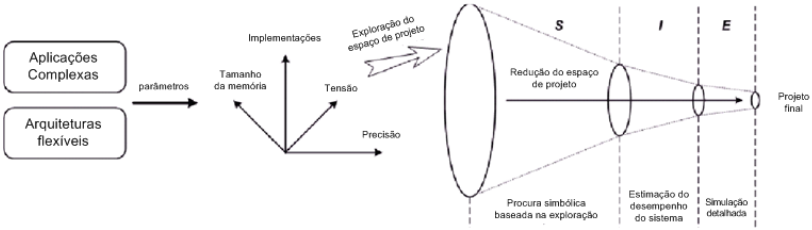


(a) Especificação do sistema embarcado (b) Uma implementação viável do sistema

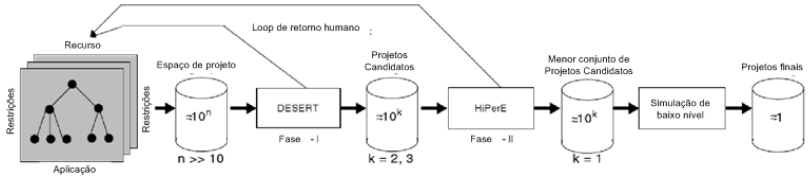
Figura 4.8: *Especificação da exploração do espaço de projeto. Apresenta um sistema embarcado e uma possível implementação*
Fonte: Adaptado de [Schlichter et al. 2006]

precisas nas primeiras etapas da DSE (passos *S, I* na figura 4.9(a)), quando há muitas soluções possíveis, e técnicas mais precisas e lentas quando o espaço de projeto for mais restrito (passo *E*), de modo que diminuem a quantidade de soluções avaliadas e também o tempo médio necessário às avaliações.

A DSE é feita em três fases, conforme apresentado na figura 4.9(b). A primeira fase basicamente elimina as soluções que violam as restrições de projeto e usa apenas formulação matemática para avaliar espaços de projeto muito grandes. Essa etapa é realizada pela ferramenta DESERT (DESERT - *DEsign Space ExploRation Tool*) e o projetista deve ajustar manualmente as restrições para que restem, ao término dessa fase, de 100 a 1000 soluções possíveis. A segunda fase utiliza um conjunto de simuladores de diferentes níveis de abstração para avaliar todas as soluções que restaram da primeira fase. A terceira fase utiliza simuladores de baixo nível para definir a solução final.



(a) Exploração hierárquica do espaço de projeto



(b) Etapas da exploração do espaço de projeto no MILAN

Figura 4.9: *Exploração do espaço de projeto e suas etapas no MILAN. A exploração é feita em três etapas distintas, sendo que em cada etapa é utilizada uma técnica de avaliação mais precisa (e mais lenta) do que na etapa anterior; entretanto, também existem menos soluções a serem avaliadas. Desse modo, soluções muito longe do ótimo são eliminadas rapidamente por técnicas menos precisas, enquanto apenas um pequeno conjunto de boas candidatas consomem mais tempo para avaliação.*
Fonte: [Mohanty et al. 2002]

PARTE III

Solução Proposta

CAPÍTULO 5

UM MODELO DE SISTEMAS EMBARCADOS DIRIGIDOS PELA APLICAÇÃO

Primeiro, resolva o problema. Depois, escreva o código.
—JOHN JOHNSON

O principal desafio do cientista da computação é não ficar confuso pela complexidade de seu próprio desenvolvimento.
—EDSGER W. DIJKSTRA

Há dois modos de construir um projeto de software. Um modo é fazê-lo tão simples que obviamente não haja deficiências. O outro modo é fazê-lo tão complicado que não haja deficiências óbvias.
—CHARLES ANTONY RICHARD HOARE

Nesta tese abordamos alguns problemas relacionados à exploração do espaço de projeto em sistemas embarcados, em especial: (1) A representação e tratamento dado a componentes de software embarcado, superficial em diversas abordagens; (2) A exploração do suporte de hardware, tanto física quanto sintetizável; e (3) A configuração e adaptação de componentes conforme o cenário de execução. Todos esses problemas refletem-se nas estruturas e operadores genéticos utilizados pelos algoritmos evolucionários, e que sofreram adaptações para suportar as soluções propostas. Todavia, esse novo modelo evolucionário, a ser apresentado no capítulo 6, por si, não representa uma solução. São necessários novos modelos e metamodelos que representem o projeto de um sistema embarcado dirigido pela aplicação, suas métricas de qualidade (ou caracterizações), que podem ser usadas para exploração, e ainda os componentes que compõem os sistemas embarcados (incluindo suas dependências, mapeamentos e configurações), seu suporte de hardware, as ferramentas utilizadas para transformar o projeto e os componentes num sistema embarcado final. Esses modelos e metamodelos são descritos neste capítulo, nas seções 5.1, 5.2 e 5.3, respectivamente. Para serem efetivamente utilizados na exploração do espaço de projeto em sistemas embarcados, eles foram integrados à suite de ferramentas da ADESD, como é descrito na

seção 5.4. Um *framework* de otimização multiobjetivo também foi desenvolvido para suportar essa exploração, e é descrito na seção 5.5, a última deste capítulo.

5.1 Metamodelo de Projeto de Sistema Embarcado

O Metamodelo de Projeto de Sistema Embarcado representa o projeto de um sistema embarcados usando a metodologia ADESD, e armazena informações sobre os requisitos e especificações de entrada do projeto, e também sobre os resultados e saídas finais do mesmo. Com ele, forma-se uma base de conhecimento sobre os projetos já realizados, o que permite que futuras explorações possam beneficiar-se de boas soluções já encontradas em projetos anteriores. A figura 5.1 apresenta a visão geral desse metamodelo.

O projeto em si é representado pela classe `SystemDesign`, que é a classe principal desse metamodelo. Um projeto relaciona-se a outros projetos similares, como projetos de diferentes equipamentos da mesma família de equipamentos, o que permite que projetos correlacionados compartilhem diferentes informações. Esse relacionamento entre projetos permite que as melhores soluções encontradas após a exploração do espaço de projeto de um deles sejam usadas como parte da população inicial para outro, melhorando a convergência de novas otimizações, inclusive para o mesmo projeto. Também permite que os mesmos suportes de hardware utilizados num projeto sejam reutilizados noutro projeto, como proposto na PbD, ou que apenas alguns componentes do suporte de hardware (ou de software) que interagem entre si e formam um "*template*", sejam compartilhados entre os projetos relacionados. Além disso, quaisquer preferências manuais especificadas pelo projetista num projeto, como um parâmetro específico para um componente, ou determinado particionamento hardware/software, por exemplo, também podem ser consideradas noutro projeto relacionado. Um `SystemDesign` é caracterizado por suas entradas e saídas. As principais entradas de um projeto de sistema embarcado dirigido pela aplicação são cinco, descritas a seguir.

Primeiro, Um conjunto de aplicações para as quais o sistema deve ser projetado. Cada aplicação é representada por uma classe `LocalFile` , que se refere ao arquivo no qual a aplicação é especificada, seja por seu código-fonte, modelo UML ou outra estrutura qualquer. Atualmente, apenas a especificação da aplicação por código-fonte em C++ está implementada. Os requisitos da aplicação são

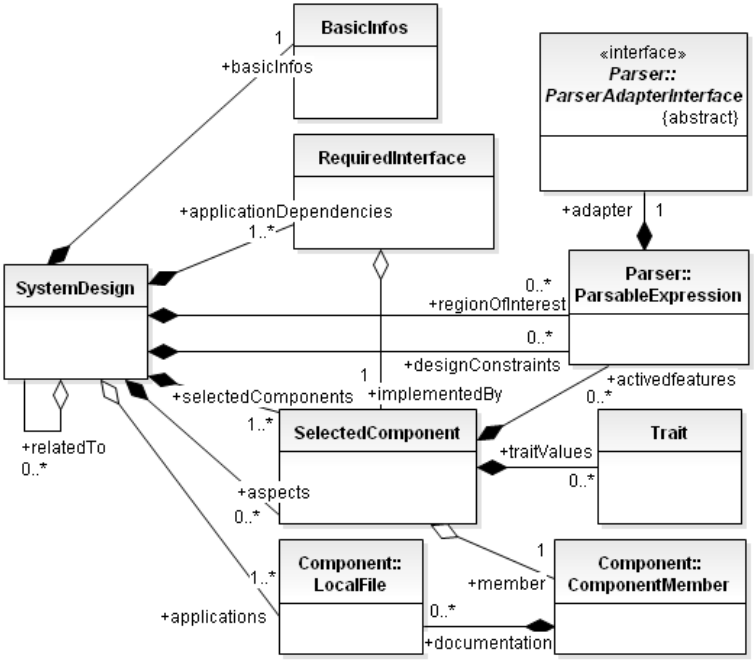


Figura 5.1: *Metamodelo de sistema embarcado dirigido pela aplicação. Um projeto de sistema embarcado é caracterizado por suas entradas (aplicações embarcadas, interfaces de componentes solicitadas pelas aplicações, restrições de projeto, regiões de interesse do projetista e aspectos a serem aplicados) e por suas saídas (conjunto de componentes selecionados e suas configurações, suporte de hardware selecionado e conjunto das melhores soluções encontradas).*

extraídos diretamente dessa especificação, como foi descrito na seção 4.1.3.

Segundo, um conjunto de interfaces de componentes requisitadas pelas aplicações, em que a interface de cada componente é representada por uma instância da classe `RequiredInterface` . Essa classe não representa um componente em si, mas a interface exigida de um componente pela aplicação, e que podem ser implementadas por diferentes componentes (agregação `implementedBy`). Como exemplo, considere a aplicação apresen-

tada na seção 4.1.3 (página 80). A aplicação tem como requisitos apenas as interfaces `void Semaphore(int)`, `void P()`, e `void V()`, que podem ser satisfeitas por um componente da classe `Semaphore`. A classe `RequiredInterface` representa esses requisitos das interfaces. Conforme apresentado na figura 4.4, essas dependências podem ser satisfeitas por mais de um componente e que, nesse exemplo, são `Semaphore_sw` ou `Semaphore_hw`, e que podem ter diferentes parâmetros e configurações nesse projeto específico. Assim, a cada objeto `RequiredInterface` está associado (`implementedBy`) a um objeto `SelectedComponent`, que especifica qual componente satisfez essa interface e quais suas configurações.

Terceiro, um conjunto de restrições de projeto, representadas pela classe `ParsableExpression` (composição `designConstraint`). Todas as soluções para o projeto devem atender, rigorosamente, as restrições de projeto estabelecidas. Cada restrição é basicamente uma expressão que segue uma gramática para representação de restrições, equações e métricas de qualidade do sistema . Essas restrições normalmente assumem a forma de inequações sobre objetivos de projeto, como, por exemplo: "`area+50`$\leq$`0`"[1] ou então equações envolvendo componentes, como, por exemplo: "`forall(SoftwareComponent.mappableTo[].getName())=='AVR8'`". Quando o projetista especifica restrições como essas, o mecanismo de otimização[2] avalia cada indivíduo em relação a elas, e aplica penalidades às soluções consideradas inviáveis. A segunda restrição apresentada, por exemplo, é interpretada pelo parser, que verifica se todos os componentes de software que foram incluídos a uma solução em particular estão mapeados a uma unidade de execução (`architecture`) com nome 'AVR8'. Isso equivale a especificar o tipo ou família de processadores a ser usado no suporte de hardware.

Quarto, um conjunto de regiões de interesse do projetista, representadas também pela classe `ParsableExpression` (composição `regionOfInterest`). De modo idêntico às restrições de projeto, cada região de interesse (ROI) do projetista é basicamente uma expressão que segue a mesma gramática para representação de restrições. As regiões de interesse também costumam assumir a forma de inequações ou equações sobre objetivos do projeto, como, por exemplo: "`consumo_energia==20`".

[1] neste caso, "`area`" deve ter sido definida no modelo de caracterização de objetivos, que é apresentado na seção 5.2.

[2] mais especificamente, a classe `Evaluator`, descrita na seção 6.3.

Diferente das restrições de projeto, as soluções não precisam seguir as ROI especificadas. Elas apenas são usadas para guiar o processo de otimização na busca de soluções próximas dessas regiões, conforme apresentado na seção 2.2.3. Quando definidas, as regiões de interesse fazem com que a otimização multiobjetivo não busque gerar soluções igualmente distribuídas sobre a fronteira de Pareto, mas sim mais concentradas sobre as regiões especificadas. As ROI especificadas podem ser utilizadas de diferentes maneiras. Atualmente elas são usadas na geração de uma distribuição de probabilidade que é aplicada ao indicador de hipervolume ponderado. Quando utilizado também como operador de seleção, essa ponderação diferenciada (distribuição de probabilidade) atua como preferência do projetista, formando um *bias* que guia a otimização para as ROI.

A quinta e última entrada para um projeto é a especificação de um conjunto de aspectos[3] a serem aplicados sobre os objetos do sistema. A representação explícita dos aspectos é necessária pois alguns deles não podem ser inferidos automaticamente a partir da aplicação[4]. Aspectos são também instâncias da classe `SelectedComponent` (composição `aspects`), já que na ADESD, aspectos são modelados como componentes que são aplicados a outros componentes através dos *adaptadores de cenários*.

Com base nessas entradas, a suite de ferramentas da ADESD guia o projetista do desenvolvimento e geração do sistema embarcado final. Depois de cada projeto concluído, seus resultados principais são mantidos neste metamodelo, definindo assim duas saídas, descritas a seguir.

Primeiro, uma relação de componentes selecionados para compor o sistema final e suas configurações. Cada componente selecionado é representado por uma instância da classe `SelectedComponent` , que faz referência a um componente específico do repositório de componentes via relação de agregação `member`. Além de especificar qual componente foi usado, ele representa também suas configurações usando a classe `Trait` (composição `traitValues`) e uma relação de funcionalidades ativadas , . A seção 5.3 trata do metamodelo de componentes e explica em detalhes o que represen-

[3] conforme definido pela programação orientada a aspectos – AOP.

[4] os aspectos não podem ser inferidos automaticamente a partir do código-fonte da aplicação, que não deve conter construções especiais para representá-los. Entretanto, os aspectos a serem "ligados" podem ser representados explicitamente em outras formas de especificação da aplicação, como modelos UML, por exemplo. Mesmo assim, por generalidade, os aspectos "ligados" são incluídos explicitamente no metamodelo de projeto.

tam *traits* e *configurable features*, já apresentadas também na seção 4.1.3.

Segundo, um conjunto das melhores soluções encontradas para esse projeto. Essas soluções, representadas por instâncias da classe `Individual`, correspondem ao resultado do processo de otimização multi-objetivo realizado na etapa de exploração do espaço de projeto e são *indivíduos*[5] completos, com genótipo, fenótipo e resultado das funções-objetivos. Em princípio, para conhecer as melhores soluções, sua representação poderia apenas incluir os valores das funções-objetivo obtidas em cada solução[6]. Porém, apenas esses valores não são de muita utilidade ao projetista. Uma vez que o projeto de um sistema embarcado foi feito (o que demanda um tempo considerável) e uma solução foi escolhida dentre um conjunto de soluções, o projetista pode ter interesse em mudar sua decisão e escolher outra solução sem ter que realizar todo o processo de projeto novamente. Para isso, é necessário que as melhores soluções já encontradas incluam também o fenótipo do indivíduo, que representa exatamente a primeira saída do projeto: um conjunto de componentes selecionados e suas configurações. Desse modo, diferentes soluções podem ser testadas sem a necessidade de refazer a exploração. Além disso, embora essas soluções sejam válidas exclusivamente para esse projeto, elas podem representar soluções aproximadas e relativamente boas para compor a população inicial de outros projetos que sejam similares ou relacionados a esse projeto de alguma forma. Assim, a realização de futuros projetos relacionados não precisaria iniciar a etapa de exploração do espaço de projeto com uma população totalmente aleatória (principalmente pela grande quantidade de soluções inviáveis que são geradas), pois as melhores soluções para o projeto atual poderiam compor pelo menos parte da população inicial para esse novo projeto, melhorando sua convergência. Para isso, o conjunto das melhores soluções precisa também incluir o genótipo dos indivíduos, de modo que optou-se representar as melhores soluções por indivíduos completos (classe `Individual`).

[5] no contexto de algoritmos evolucionários.

[6] esses valores são obtidos com a avaliação do fenótipo dos indivíduos no contexto do projeto de sistemas embarcados.

5.2 Metamodelo de Caracterização de Custos do Sistema

A exploração do espaço de projeto visa encontrar as melhores soluções (sistemas embarcados) que minimizam um conjunto de funções-objetivo, que representam os "custos" do sistema. Entretanto, é comum nesse domínio que as expressões matemáticas de tais funções-objetivo sejam desconhecidas, de modo que as soluções são avaliadas através de co-simuladores, modelos analíticos ou outras técnicas de obtenção de estimativas para os valores dessas funções-objetivos para a solução sendo avaliada. Co-simuladoes de baixo nível são muito lentos, de forma que modelos mais rápidos e menos precisos são usados para estimar os valores dessas funções-objetivo desconhecidas, pelo menos quando ainda há uma grande quantidade de possíveis soluções. Esses modelos podem ser regressões lineares ou polinomiais, lógica difusa ou redes neurais, entre outras alternativas encontradas na literatura. Todavia, muitas dessas técnicas restringem o a forma de representação das funções (que são sempre polinômios, combinações lineares, etc) ou são estáticos, ou seja, determina-se seus coeficientes ou outros parâmetros a partir de uma fase de treinamento e depois utiliza-se o modelo para estimar os valores das funções-objetivo (ou custos do sistema) para futuros sistemas. O metamodelo de caracterização de objetivos de otimização flexibiliza a forma de representação das funções-objetivo, ao permitir a mudança da própria equação em si[7] e também evolui suas estimativas, ao evoluir a equação cada vez que um novo sistema é projetado.

A visão geral desse metamodelo é apresentada na figura 5.2, que representa cada caracterização (classe `Characterization`) dos custos dos sistema. Uma *caracterização de custo do sistema* é uma característica do sistema sendo projetado e que pode ser utilizada como função-objetivo a ser otimizada, como área, energia ou latência, que são casos comuns, embora muitos outros possam ser utilizados. Os custos normalmente utilizados para exploração do espaço de projeto dependem de diversos fatores, que são outros atributos/características (classe `ComponentAttribute`) de componentes. Por exemplo: o custo de memória de um componente de software dependende do conjunto de instruções da arquitetura ao qual está mapeado,

[7] por exemplo, uma equação representada pela *String* "y=2.45*x1^5-0.0035*x2^2" pode evoluir, usando Programação da Expressão Genética (GEP), para equações totalmente diferentes, como "y=2.52*cos(x1-0.0035*x2) + cos(x1)/sqrt(123.4*x2)", por exemplo, através de seus operadores de variação.

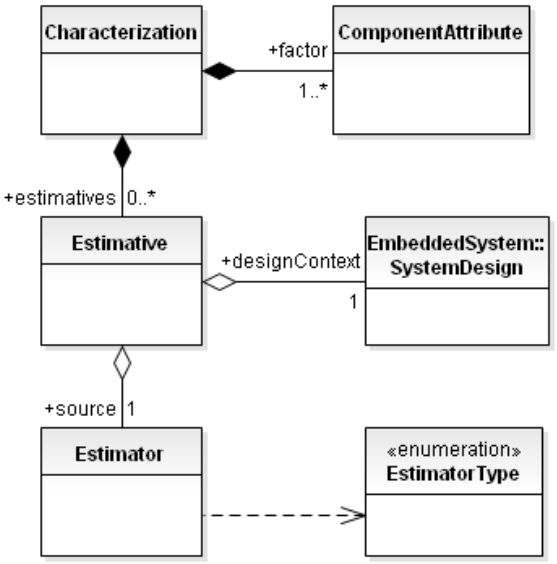


Figura 5.2: *Metamodelo de caracterização de objetivos de otimização. Cada caracterização que pode ser utilizada para como resultado de uma função-objetivo para otimização é influenciada por fatores que correspondem a características dos componentes, e pode ser estimada usando diferentes estimadores, sempre com base num contexto, que é o projeto de sistema embarcado no qual ela foi utilizada.*

entre outros fatores. Porém, é impraticável incluir todas as características de todos os componentes como fatores (variáveis) de uma função-objetivo, e também mostrou-se ineficiente a identificação automática dessas variáveis[8]. Assim, é responsabilidade do projetista especificar, para cada caracterização de custo do sistema, quais são os fatores que ele considera que tenham maior influência sobre ela. Para cada *caracterização de custo* podemos ter diferentes estimativas (associação `estimatives`) de seu valor, dependendo do valor dos fatores que a influenciam. Como o valor dos fatores depende do projeto de sistema embarcado (`SystemDesign`) específico sendo realizado, ele

[8]Mesmo com o uso de técnicas estatísticas e de projeto de experimentos, testados por este autor, a existência de muitas variáveis (centenas ou milhares) inviabiliza a identificação das características com maior influência sobre a função-objetivo.

corresponde ao contexto no qual a estimativa foi realizada e é válida. Cada estimativa pode ser obtida a partir de diferentes estimadores (`Estimator`), como simuladores analíticos, co-simuladores em nível RTL ou mesmo o *feedback* do projetista a partir de sistemas já projetados. Cada possível estimador possui uma precisão diferente. Portanto, há estimativas melhores que outras.

Os diferentes estimadores, em geral, produzem um único número real que representa a caracterização de custo do sistema sendo estimado, que é armazenado na classe `Estimative`. Para permitir a criação e evolução contínua de estimadores baseados em modelos analíticos, além do valor numérico estimado, esse metamodelo armazena uma expressão matemática que o representa, ou seja, a avaliação dessa expressão, no contexto especificado[9], produz um valor numérico próximo ao valor real ou estimado. Essa expressão matemática, que forma o modelo analítico para futuras estimativas dessa caracterização do sistema, assume a forma de uma *String* que precisa ser avaliada por um interpretador, e segue a linguagem Karva ou P-GET (ver seção 3.2.2), ou seja, corresponde a um indivíduo GEP. Essa expressão é adaptada pelo mecanismo de otimização multiobjetivo utilizado pela técnica de evolução de expressões matemáticas existente na GEP. O próprio "histórico" (conjunto associado) de estimativas é utilizado como população inicial para essa adaptação a novos projetos, de modo que as estimativas tendem a tornar-se melhores à medida que mais projetos são feitos ou soluções avaliadas, e a expressão evolui com dois objetivos: (1) Minimzar a diferença quadrática entre os valores reais/estimados do sistema e os valores produzidos a partir da expressão sendo otimizada; e (2) Minimizar o tamanho da *String* que codifica a expressão endo otimizada. A evolução das equações que representam uma caracterização do sistema que pode ser objetivo de otimização corresponde, por si, num problema de otimização multiobjetivo, formalmente definido na equação (5.1), onde n é a quantidade de pontos conhecidos, $e(i)$ é o valor real ou uma estimativa do valor real de uma característica do sistema, $\hat{e}(i)$ é o valor obtido a partir da interpretação (*parsing*) da *String* da equação sendo otimizada, e $\mid \hat{e} \mid$ representa o comprimento da *String* da equação.

$$\begin{aligned} Minimizar : f_1 &= \sum_{i=1}^{n} (e(i) - \hat{e}(i))^2 \\ f_2 &= \mid \hat{e} \mid \end{aligned} \tag{5.1}$$

[9] um projeto de sistema embarcado com valores específicos dos fatores que influenciam a caracterização de custo do sistema sendo estimada.

Desse modo, distingue-se dois usos para a otimização multiobjetivo na exploração do espaço de projeto em sistemas embarcados: Inicialmente, e mais óbvia, a otimização do problema de exploração em si, evoluindo soluções que representam sistemas embarcados que sejam ótimos em relação aos objetivos sendo minimizados; objetivos esses que correspondem a caracterizações do sistema embarcado (área, energia, preço, tamanho, etc). A segunda utilização corresponde especificamente à utilização do metamodelo de caracterização de objetivos de otimização e uma contribuição desta tese, qual seja, a evolução das próprias equações que representam as funções-objetivo sendo otimizadas e, com isso, a exploração de novos espaços de projeto, representados por funções-objetivo desconhecidas, e cujos estimadores não precisam existir *a priori*, mas evoluem à medida que novos sistemas embarcados vão sendo projetados. Embora o funcional, a avaliação desse metamodelo depende de uma boa quantidade de sistemas embarcados já projetados e de muitas estimativas e, portanto, foge do escopo desta tese.

5.3 Metamodelo de Componentes Embarcados

O "Metamodelo de Projeto de Sistema Embarcado", apresentado na seção 5.1, inclui a representação dos componentes utilizados nesse sistema. Os componentes que podem fazer parte de um sistema embarcado são representados no "Metamodelo de Componentes Embarcados", descrito nesta seção. Ele define as características que permitem a seleção dos componentes que satisfazem os requisitos da aplicação, a configuração e adaptação desses componentes, a compilação dos componentes de software, a síntese dos componentes de hardware, e a caracterização de cada tipo de componente, permitindo a exploração de diferentes espaços de projeto. A visão geral de pacotes desse metamodelo é apresentada na figura 5.3. Os pacotes foram organizados em função da interface dos componentes, já que na ADESD os componentes são agrupados em famílias, e todos os componentes membros de uma família são vistos como um "super-componente" [Fröhlich 2001] com uma interface única, a *interface inflada*. Na figura 5.3 percebe-se que o pacote-base é o pacote `Component`, que contém os elementos básicos de todos os componentes, incluindo sua estruturação em famílias, dependências e confi-

gurações, entre outros. Basicamente, componentes podem ser físicos (pacote `PhysicalComponent`) ou lógicos (pacote `CyberComponent`), aproximando a ADESD da abordagem de CPS (CPS - *Cyber Physical Systems*).

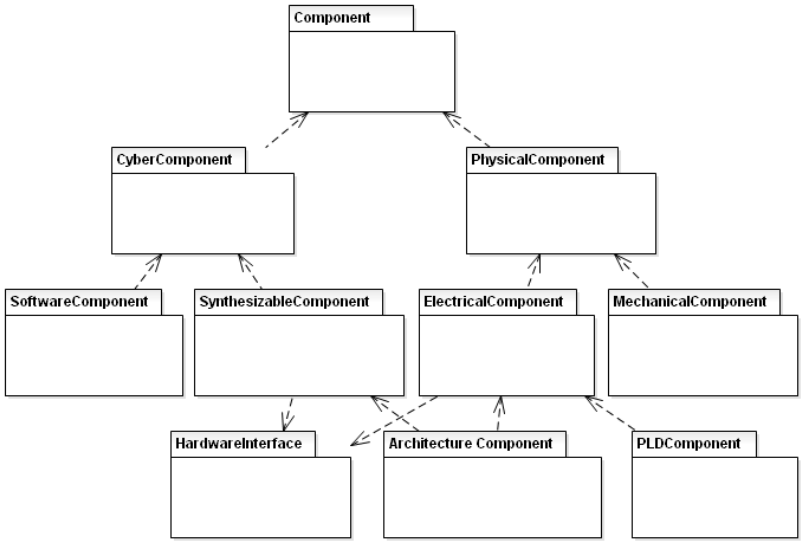


Figura 5.3: *Estrutura geral do metamodelo de componentes. Componentes podem ser lógicos (software, hardware sintetizável) ou físicos (elétricos ou mecânicos). Elementos processadores podem ser físicos ou sintetizáveis, e PLDs são componentes físicos aos quais hardware sintetizáveis devem ser mapeados.*

Componentes físicos correspondem a componentes mecânicos ou componentes eletro-eletrônicos. Nesta tese, a ênfase nos componentes físicos é dada aos componentes eletro-eletrônicos, como chips, conectores, baterias, elementos processadores (pacote `ArchitectureComponent`), dispositivos lógicos programáveis (pacote `PLDComponent`), entre outros elementos físicos do hardware, que possuem interface de hardware (pacote `HardwareInterface`). Destaque é dado às arquiteturas (processadores, possivelmente associados a outros componentes, como memória e dispositivos periféricos) e aos dispositivos lógicos programáveis, pois os componentes lógicos devem ser mapeados a eles. Componentes lógicos correspondem aos elementos não físicos do sistema e que podem ser, basicamente,

software (pacote `SoftwareComponent`) e hardware sintetizável (pacote `SynthesizableComponent`). Componentes de software são mapeados a arquiteturas, enquanto componentes de hardware sintetizável são mapeados a dispositivos lógicos programáveis (PLD). Esse metamodelo também representa tanto arquiteturas físicas (como microcontroladores) quanto arquiteturas sintetizáveis (*softcores*). Nesse último caso, a arquitetura também é um componente lógico sintetizável que deve ser mapeado a um PLD.

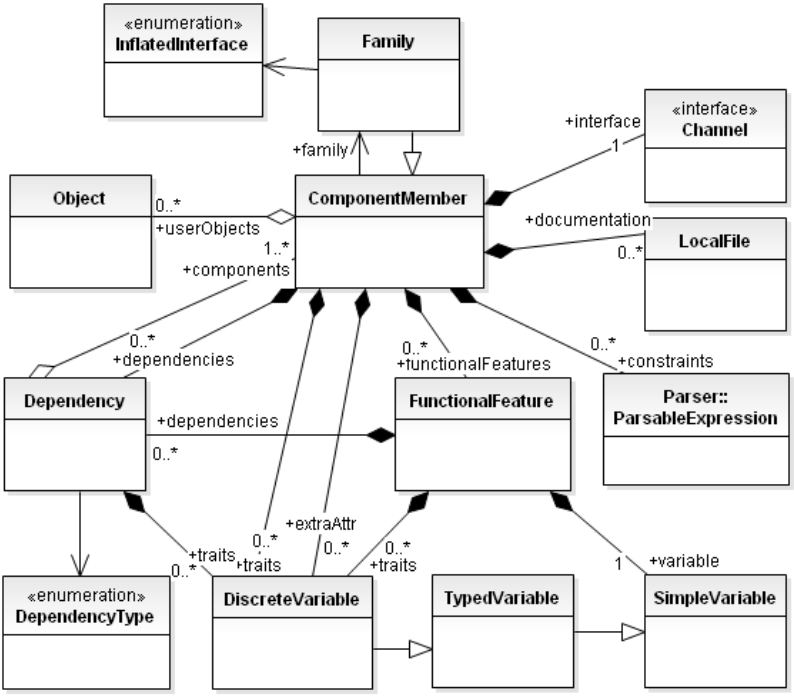


Figura 5.4: *Modelo de um componente-base. Cada componente pertence a uma família de componentes, possui traits (configurações), dependências de outros componentes ou famílias, e características configuráveis, ambas com suas próprias configurações.*

O pacote `Component`, que representa as características presentes em todos os componentes que podem compor um sistema embarcado, é apre-

sentado na figura 5.4. A principal classe é `ComponentMember`, da qual especializam-se todos os demais tipos de componentes. Na ADESD, todos os componentes fazem parte de uma família, representada pela classe `Family`, que é vista como um "super-componente" com uma interface inflada. A classe `InflatedInterface` apresenta o tipo da interface inflada, conforme definido em [Fröhlich 2001]. Componentes podem exigir a inclusão de um ou mais componentes (ou famílias de componentes) do qual eles dependem, formando uma dependência, representada pela classe `Dependency` (composição `dependencies` do componente). Basicamente, dependências podem ser de três tipos, definidos na enumeração `DependencyType`: *Requisit*, *Alternative* e *Exclude*. Uma dependência alternativa exige que apenas um dos componentes da dependência seja escolhido, sendo que essa escolha faz parte da exploração do espaço de projeto (DSE). Componentes podem ser configurados (automática ou manualmente) através de parâmetros, denominados *traits*, cujo valor também precisa ser escolhido durante a DSE. Cada parâmetro é representado por uma instância da classe `DiscreteVariable` (composição `traits` do componente). A adaptação de um componente se dá, entre outras possibilidades, pela ativação ou não de uma característica configurável (*configurable feature*), representada pela classe `FunctionalFeature`. Uma característica configurável também pode ter seus próprios parâmetros (composição `traits` da característica configurável) e pode ainda exigir a inclusão de um ou mais componentes que implementem tal característica, formando sua própria dependência (composição `dependencies`). A ativação ou não dessa configuração, seus parâmetros e possivelmente também suas dependências alternativas precisam ser escolhidos durante a DSE. A interconexão de componentes, principalmente de hardware, exige que restrições sobre características dos componentes sejam satisfeitas. Essas restrições (composição `constraints`) podem ser representadas por expressões (`ParsableExpression`) que são interpretadas durante o processo de avaliação de uma solução para o projeto de sistemas embarcados. A interface de comunicação entre componentes é o `Channel`, e deve ser estendida pelas especializações do `ComponentMember`. Basicamente, o `Channel` corresponderá a métodos, tipos e enumerações nos componentes de software e a pinos e barramentos nos componentes de hardware.

Os componentes *cyber*/lógicos incluem software e hardware sinte-

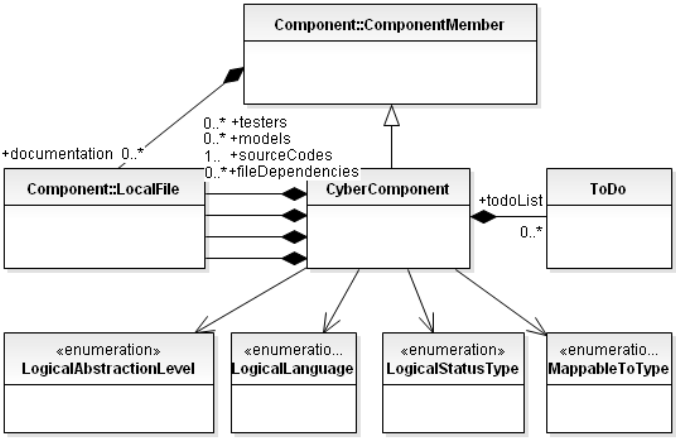


Figura 5.5: *Modelo de componente cyber/lógico. Qualquer componente cyber/lógico está associado a arquivos que representam seu código-fonte, modelos e testadores, entre outros, além de especificações de qual nível abstração, linguagem e estágio de desenvolvimento está especificado o componente.*

tizável[10]. Além das informações de qualquer componente, que permite sua adaptação e configuração, os modelos desses componentes descrevem outras informações que permitem sua compilação e síntese de forma automática. A figura 5.5 apresenta o modelo de componentes *cyber*/lógicos (classe `CyberComponent`). Qualquer componente cyber/lógico possui associações com arquivos (classe `LocalFile`), que representam seu código-fonte (composição `sourceCodes`), dependências[11], testadores, e modelos, entre outros. Também estão associados a enumerações que especificam seu nível de abstração (`LogicalAbstractionLevel`) e linguagem de especificação (`LogicalLanguage`), usados para selecionar as ferramentas de software necessárias para transformá-los até suas formas finais[12]. A enu-

[10] em princípio, podem incluir também uma representação que possa gerar software ou hardware sintetizável, como máquinas de estados finitos ou especificações em SystemC, por exemplo.

[11] dependências de outros arquivos são distintas das dependências de outros componentes e famílias, descritas para a classe `ComponentMember`.

[12] compiladores, ligadores e sintetizadores são alguns exemplos. A linguagem de descrição do componente é utilizada para selecionar a ferramenta adequada para transformá-lo.

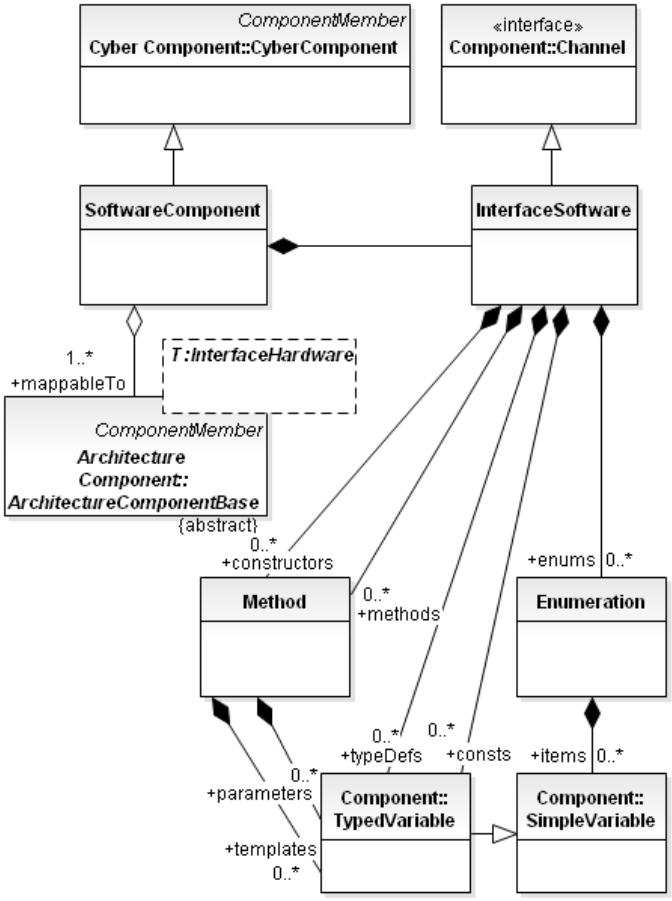


Figura 5.6: *Modelo de componente de software. Componentes de software são mapeados em uma ou mais arquiteturas para serem executados. Também possuem uma interface e podem ser acessados por seus construtores, métodos, enumerações, typedefs e constantes.*

meração `LogicalStatusType` e a classe `ToDo` permitem especificar o nível de maturidade do desenvolvimento do componente, útil na seleção de

componentes, e a enumeração `MappableToType` auxilia na especificação de mapeamentos a arquiteturas ou PLDs, nos casos em que muitos mapeamentos são possíveis.

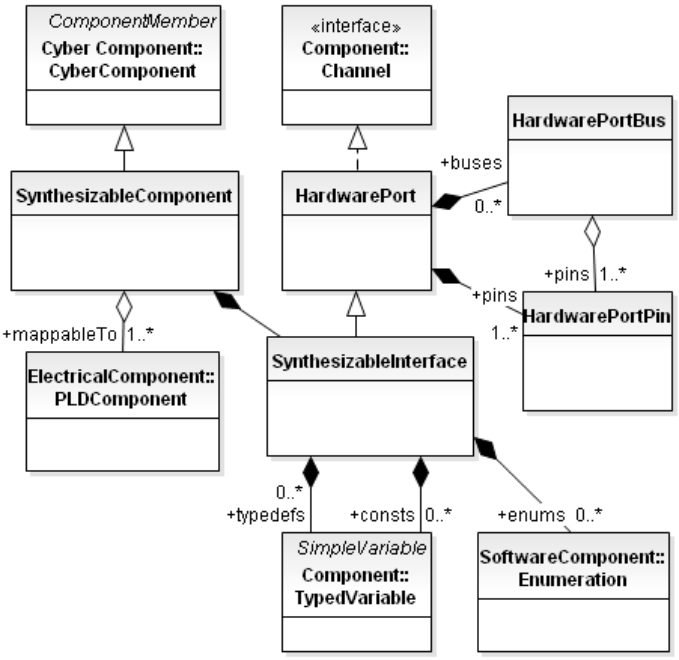


Figura 5.7: *Modelo de componente de hardware sintetizável. Componentes de software são mapeados em uma ou mais arquiteturas para serem executados. Também possuem uma interface e podem ser acessados por seus construtores, métodos, enumerações, typedefs e constantes.*

A figura 5.6 apresenta o modelo de componentes de software (`SoftwareComponent`) que, basicamente, estendem a interface básica do componente (`Channel`) para `InterfaceSoftware`, que especifica seus métodos e construtores (classe `Method`), enumerações (classe `Enumeration`), *typedefs* e constantes (composições `typeDefs` e `consts`), que são variáveis tipadas (classe `TypedVariable`). O modelo

de componentes sintetizáveis é apresentado na figura 5.7. Qualquer hardware, físico ou sintetizável, expande a classe `HardwarePort`, que define a interface desses componentes, composta por pinos (`HardwarePortPin`) e barramentos (`HardwarePortBus`). Além desses, hardware sintetizável pode definir enumerações, *typedefs* e constantes. Parametrizações (como os *generic* do VHDL) são modeladas como *traits* no componente-base.

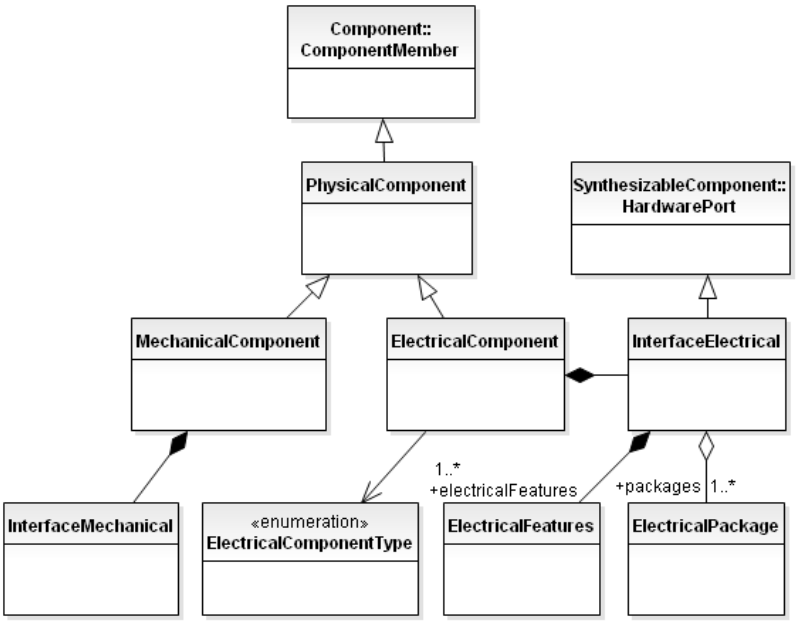


Figura 5.8: *Modelo de componente físico. Um componente físico pode ser mecânico ou eletro-eletrônico, que estende a interface de hardware acrescentando características elétricas a seus pinos e barramentos, além de encapsulamentos.*

A figura 5.8 mostra o modelo de componentes físicos, que podem ser mecânicos (`MechanicalComponent`) ou eletro-eletrônicos (`ElectricalComponent`). Componentes eletro-eletrônicos estendem a interface de hardware, incluindo características elétricas (`ElectricalFeatures`) ao componente e a cada um de seus barramentos e pinos, além de um conjunto de encapsulamentos

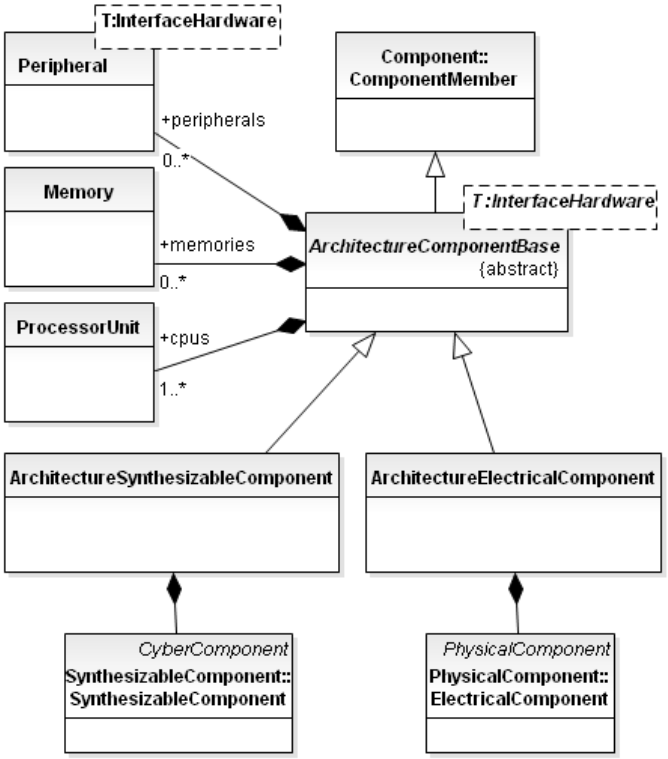


Figura 5.9: *Modelo de componente de arquitetura. Contém processadores, memórias e dispositivos periféricos. Especializando a classe básica e compondo componentes específicos, pode-se ter arquiteturas físicas ou sintetizáveis.*

`ElectricalPackage`. Entre os possíveis tipos de componentes eletro-eletrônicos (`ElectricalComponentType`), destacam-se as arquiteturas e os dispositivos lógicos programáveis, por sua relação com os componentes de software e de hardware sintetizável, respectivamente. O modelo de arquiteturas é apresentado na figura 5.9. Arquiteturas são elementos processadores, que incluem um processador (`ProcessorUnit`), como microprocessadores, e possivelmente também memórias (`Memory`) e dispositivos pe-

riféricos (`Peripheral`), como microcontroladores. Arquiteturas podem ser físicas (microprocessadores, microcontroladores) ou sintetizáveis (*softcores*), estendendo a classe `ArchitectureComponentBase<T>`, de forma a definir o tipo de sua interface (parâmetro `<T>`). Cada possível especialização de arquitetura, física (`ArchitectureElectricalComponent`) ou sintetizável (`ArchitectureSynthesizableComponent`) é composta com um componente análogo.

5.4 Integração com Suite de Ferramentas da ADESD

A suite de ferramentas da ADESD, até a versão imediatamente anterior ao desenvolvimento desta tese, teve sua estrutura e funcionamentos básicos descritos na seção 4.1.3. Basicamente, ela consistia de três etapas (e ferramentas) principais, executadas sequencialmente, num fluxo de projeto único e sequencial: `Analisador`, `Configurador` e `Gerador`. A figura 4.5, reapresentada abaixo na figura 5.10 por comodidade, ilustra esse fluxo. Até essa versão, o projetista precisava especificar a arquitetura-alvo do sistema sendo projetado e também as configurações dos componentes selecionados automaticamente pela suite de ferramentas, além de escolher, manualmente, qual componente seria utilizado, no caso mais de um componente implementar os serviços solicitados pela aplicação-alvo. Sem margem para exploração do espaço de projeto, o sistema final era gerado, e consistia da imagem do software (aplicação e software de suporte de execução) e possivelmente também o *bitstream* de configuração da FPGA quando a arquitetura-alvo era um *softcore*. Ao término da geração do sistema, as informações de projeto eram perdidas.

Com o desenvolvimento desta tese, e conforme o que foi apresentado no presente capítulo, foram criados: (1) um metamodelo de projeto de sistema embarcado, que representa um projeto segundo a ADESD, incluindo suas entradas e saídas, de modo a formar uma base de conhecimento para futuros projetos e projetos de famílias de produtos; (2) um metamodelo de caracterização de objetivos de otimização, que formam uma base de conhecimento sobre as estimativas obtidas para cada característica que pode ser explorada (otimizada), e permite a evolução da expressão matemática que representa essa característica, permitindo refinar os estimadores utilizados para avaliar

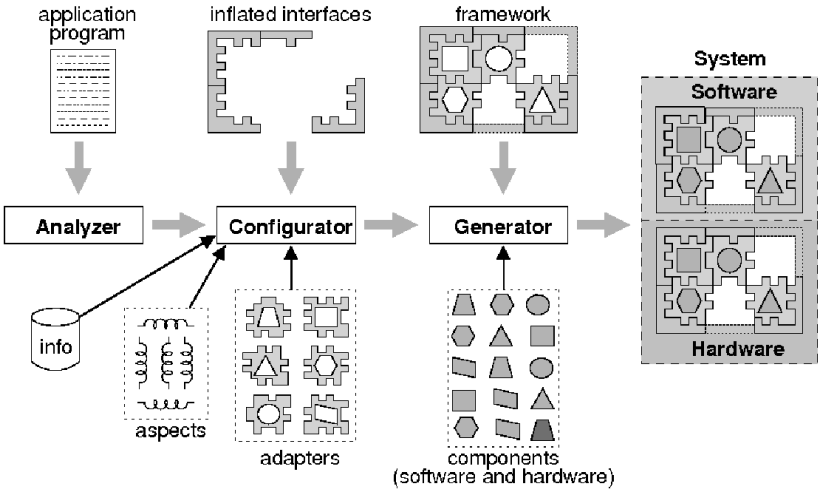


Figura 5.10: *Processo de geração de sistemas embarcados pela ADESD. (REAPRESENTAÇÃO DA FIGURA 4.5) O Analisador extrai as interfaces infladas da aplicação, enquanto o Configurador escolhe os componentes que satisfazem tais interfaces, e os configura. Por fim, o Gerador utiliza ferramentas de terceiros para compilar o software e sintetizar o hardware, gerando o sistema embarcado final.*
Fonte: [Fröhlich 2001]

cada solução alternativa, à medida que novas estimativas e projetos vão sendo feitos; (3) um modelo de componente, que representa um componente lógico ou físico, de software ou de hardware, que pode compor um sistema embarcado dirigido pela aplicação; e (4) um modelo evolucionário para exploração do espaço de projeto, que mapeia os outros modelos e metamodelos, e que foi incluído num MOP (MOP - *MultiObjective Problem*), passível de otimização pelo *framework* de otimização desenvolvido (a ser apresentado no capítulo 6). Com isso, a estrutura da nova suite de ferramentas da ADESD para a ser a apresentada na figura 5.11. Os pacotes destacados correspondem àqueles em que houve maior desenvolvimento ou contribuição com esta tese.

Na nova versão da suite de ferramentas, ilustrada na figura 5.11, o `Analisador` ainda é o responsável por extrair os requisitos da aplicação, definidas como um conjunto de interfaces de componentes que representam serviços a serem executados. Dependências são resolvidas e todos os com-

ponentes recursivamente necessários são incluídos numa possível solução. Contudo, dependências alternativas, especificação das configurações (*traits*) e mapeamento à arquitetura-alvo não precisam mais ser resolvidas manualmente pelo projetista. O processo de exploração do espaço de projeto, representado pela ferramenta `Explorer` e cuja infraestrutura é baseada no mecanismo de otimização multiobjetivo com algoritmos evolucionários, é responsável por gerar diferentes soluções alternativas, avaliá-las a selecionar iterativamente as melhores soluções. Após o projetista escolher, dentre um conjunto limitado de melhores soluções, a que lhe interessa, o `Gerador` é utilizado para gerar a imagem final do software, possivelmente o *bitstream* de configuração da FPGA (caso a solução envolva hardware sintetizável), e também uma versão preliminar do esquemático do suporte de hardware físico, que é uma contribuição desta tese.

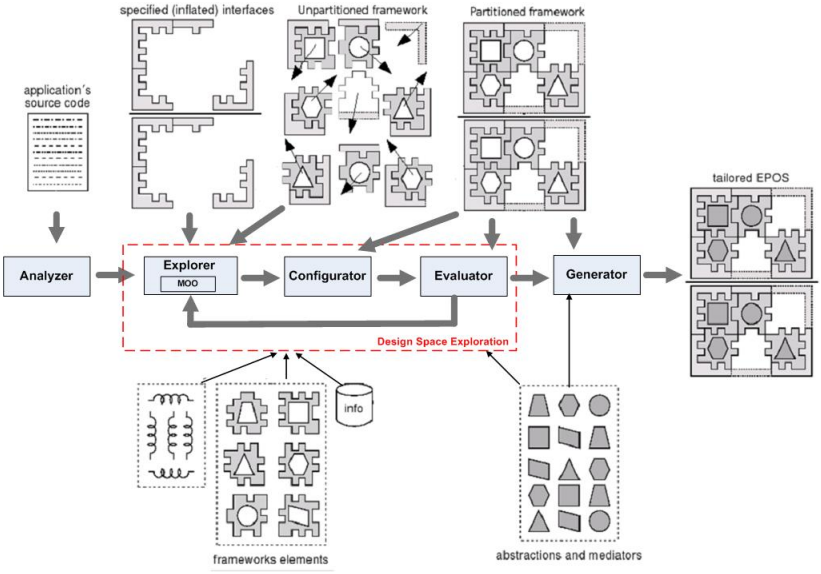


Figura 5.11: *Novo processo de projeto de sistemas embarcados com a ADESD. Com o desenvolvimento desta tese, o fluxo de projeto tornou-se iterativo, com o processo de exploração em seu cerne, em que o* `Explorer` *gera diferentes soluções alternativas, que são configuradas pelo* `Configurador` *e avaliadas pelo* `Estimador`*. Quando a solução final é encontrada, o sistema é gerado pelo* `Gerador`*.*

O modelo da nova suite de ferramentas da ADESD é apresentado na figura 5.12, e é composto de quatro pacotes principais, descritos a seguir.

O pacote `DesignFlow` representa o fluxo de projeto a ser aplicado num projeto específico e as ferramentas que executam cada etapa desse fluxo. Com isso, sequências distintas de invocação de ferramentas poderiam ser aplicadas para cada projeto, desde que a integração entre entradas e saídas de cada ferramenta seja compatível (o que não é garantido pelo sistema desenvolvido). De modo geral, o fluxo de projeto adotado é aquele apresentado na figura 5.11. Os principais desenvolvimentos nesta tese incluem principalmente os pacotes `Explorer` e `Estimator`. O primeiro é um adaptador para mecanismos de exploração do espaço de projeto (e que utiliza a otimização multiobjetivo disponibilizada pelo pacote `MOO`, descrito posteriormente), enquanto o segundo é um adaptador para diferentes mecanismos de obtenção de estimativas dos custos de cada solução, ou seja, avaliação da aptidão dos indivíduos (soluções) gerados pelo algoritmo evolucionário.

O pacote `ToolManager` faz o gerenciamento do suite de ferramentas em geral, e o principal desenvolvimento está associado principalmente ao metamodelo de projeto de sistema embarcado que foi associado ao pacote `EmbeddedSystemDesign`. Ele é responsável por manter uma base de informações sobre os projetos já realizados, associar o contexto de um projeto a estimativas obtidas para melhorar os modelos de obtenção de estimativas (`Estimator`) e reusar boas soluções já encontradas em projetos anteriores na exploração de soluções para os novos projetos.

O pacote `Repository` modela e armazena os componentes reusáveis que podem ser compostos para gerar um novo sistema embarcado. Ele inclui o metamodelo de componentes embarcados, associado ao pacote `Component`, o metamodelo de caracterização de objetivos de otimização, associado ao pacote `SystemQualityMetric` e outros metamodelos menos importantes que não foram previamente descritos. Em especial, o repositório modela e armazena também agrupamentos de componentes, denominados nesta tese como "*templates*", e que representam estruturas comuns a vários projetos, como sub-circuitos ou mesmo suportes completos de hardware, nos casos mais usuais. Uma relação dos componentes atualmente cadastrados na suite de ferramentas da ADESD pode ser encontrada no apêndice B.

Por fim, o pacote `MOO` representa um *framework* para otimização mul-

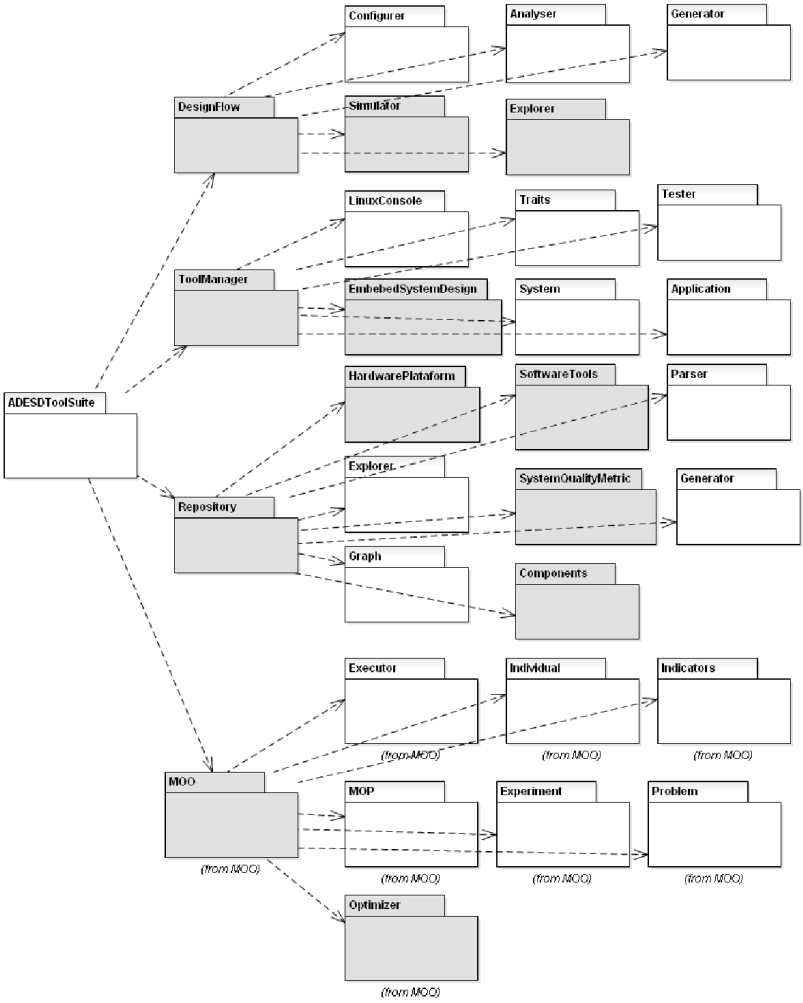


Figura 5.12: *Modelo da nova suite de ferramentas da ADESD. Os modelos e metamodelos desenvolvidos foram integrados à suite de ferramentas, e os principais pacotes alterados aparecem destacados.*

tiobjetivo (MOO - *MultiObjective Optimization*), e é utilizado com mecanismo de exploração do espaço de projeto. Por representar a etapa do projeto que é o foco desta tese, ele é descrito em mais detalhes na próxima seção.

5.5 Framework para Otimização Multiojetivo

A suite de ferramentas da ADESD, apresentada na seção anterior, inclui um *framework* de otimização multiobjetivo que foi desenvolvido para permitir a exploração do espaço de projeto de sistemas embarcados. Esta seção visa descrever aspectos gerais do modelo e estrutura desse *framework*. Vários *frameworks* de otimização são encontrados na literatura e poderiam ter sido usados. De fato, o *framework* desenvolvido baseia-se em (e utiliza parte do código de) dois deles: O PISA [Bleuler et al. 2003], do Instituto Federal de Tecnologia de Zurich, e o OPT4J [Lukasiewycz, Glaß e Reimann 2010].

Para ser possível variar pequenos componentes dos modelos evolucionários, como o tipo de gene, o tipo de dominância, os indicadores de qualidade, e os operadores utilizados, entre outros, foi necessária uma granularidade mais fina na decomposição do otimizador do que aquela originalmente provida pelo PISA e pelo OPT4J. Também foi automatizado o processo de geração de diferentes experimentos e de sua avaliação, feita a integração de códigos de diferentes otimizadores, escritos em diferentes linguagens de programação e sua integração à suite de ferramentas da ADESD. O modelo simplificado do *framework* de otimização desenvolvido é apresentado na figura 5.13, em que `MOO` (*MultiObjective Optimization*) é o pacote principal e compõe-se dos demais pacotes apresentados nessa figura. Novamente, os pacotes destacados correspondem àqueles em que houve maior desenvolvimento ou contribuição com esta tese.

Toda otimização busca resolver um problema, representado pelas classes do pacote `Problem`. Cada problema precisa ser codificado num genótipo, que é traduzido para um fenótipo e avaliado no contexto desse problema durante a otimização. Os pacotes `Creator`, `Translator` e `Evaluator`, respectivamente, têm essas funções. Para a avaliação também é necessário especificar o que é uma solução preferível, com base em algum tipo de dominância (não necessariamente Pareto), que pode ser representada pelo pacote `Dominance`. Diferentes problemas

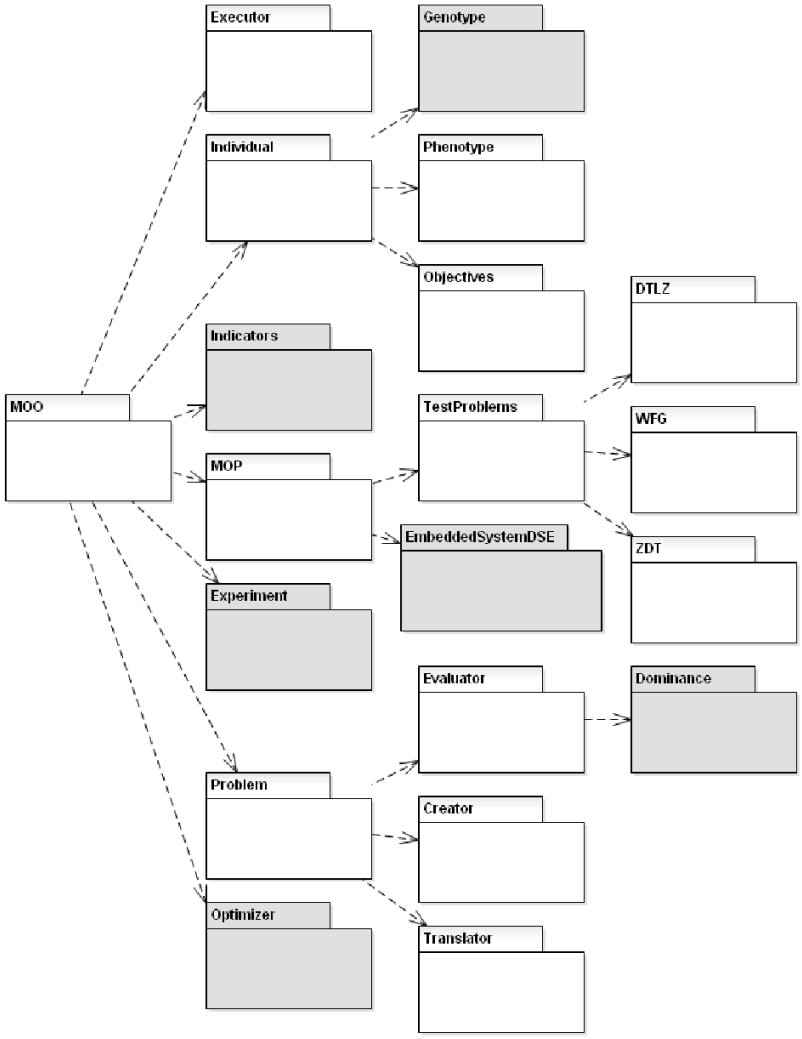


Figura 5.13: *Modelo do framework de otimização multiobjetivo. Apresenta o pacote principal* `MOO` *(Multiobjective Optimizator) e seus pacotes associados.*

foram modelados, e compõem o pacote `MOP` (*Multiobjective Problems*), sendo que apenas os principais estão representados na figura: `DTLZ`, `ZDT`, `WFG` e `EmbeddedSystemDSE`. Possíveis soluções do problema são representados como indivíduos (`Individual`), compostos por um genótipo (`Genotype`), um fenótipo (`Phenotype`) e objetivos de otimização (`Objective`). O responsável pela evolução e otimização, propriamente ditas, são as classes do pacote `Optimizer` e que podem representar diferentes tipos de otimização. O controle da execução do otimizador é de responsabilidade do pacote `Executer` e os resultados da otimização são avaliados por um indicador de qualidade (`Indicator`). Para uma otimização específica, o projetista precisa escolher, basicamente, as classes dos pacotes `Problem`, `Optimizer` e `Indicator` que irão compor o `MOO`. O pacote `Experiment` permite a mudança automática da escolha dessas classes, gerando diferentes conjuntos de experimentos que são executados automaticamente.

Por sua complexidade, a visão geral do modelo do otimizador (`Optimizer`) é apresentado separadamente, na figura 5.14. Um otimizador precisa, inicialmente, montar o genótipo da população inicial, o que é feito pelas classes do pacote `IndividualBuilder`, que utilizam o `Creator` especificado pelo problema. `Completer` usa as demais classes de `Problem` para avaliar a aptidão de cada solução[13]. Em seguida um algoritmo de seleção (`Selector`), como SPEA, NSGA2 ou MOEA/D é utilizado para escolher as melhores soluções (não-dominadas) da geração atual, que podem ser armazenadas numa lista elitista das melhores soluções encontradas em todas as gerações (`Archive`), que são as soluções aproximadas do conjunto de Pareto e correspondem ao resultado final da otimização. O pacote `Coupler` escolhe soluções para formar indivíduos-pais, conforme um algoritmo específico[14]. Os indivíduos-pais são, então, recombinados por `Mating` para formar indivíduos-filhos. De fato, qualquer operador genético binário (ou n-ário, dependendo da quantidade de indivíduos-pais) pode ser usado para "recombiná-los". Os indivíduos-filhos sofrem variações (`variation`) através de diferentes operadores, como a mutação (`Mutate`) ou a regulação

[13]Embora uma solução tenha sido denominada `Individual`, ela representa um indivíduo apenas num algoritmo evolucionário, mas pode representar partículas ou outros modelos de solução, dependendo do otimizador.

[14]A escolha dos indivíduos-pais pode considerar nichos, subpopulações, diversidade dos indivíduos, migrações ou ainda outros aspectos evolucionários.

da expressão genética `ExpressedRegulator`, desenvolvido nesta tese. Cada operador genético é parametrizado (`Parameter`) de diferentes formas, das quais a utilização (e variação) de diferentes distribuições de probabilidade (`Distribution`) é a mais comum. Informações diversas sobre a otimização, como a aptidão dos indivíduos, diversidade da população e outras, podem ser obtidas ao compor coletores de dados (`DataLogger`) ao otimizador.

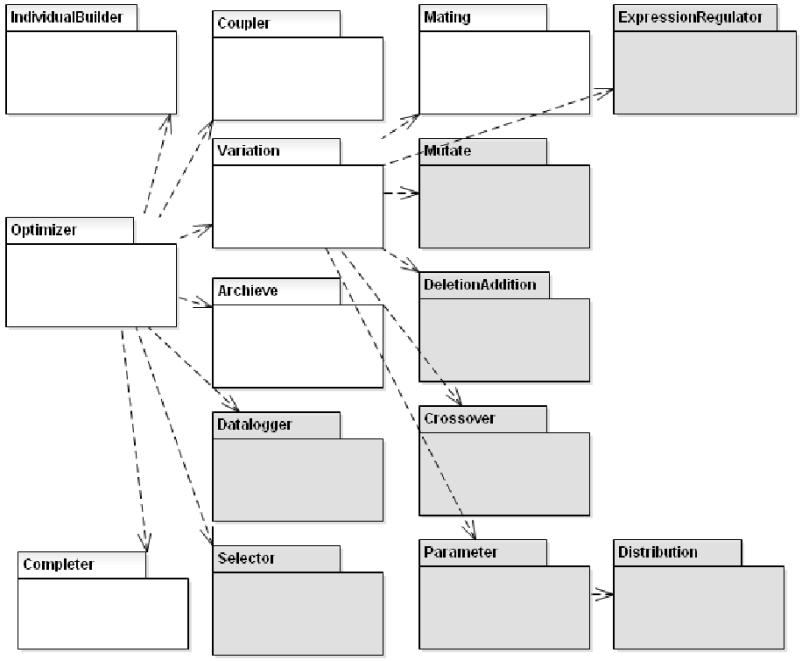


Figura 5.14: *Modelo do otimizador. Mostra o pacote principal* `Optimizer` *e seus pacotes associados.*

Esta seção apresentou a estrutura geral do *framework* de otimização que foi desenvolvido principalmente a partir do *framework* OPT4J, destacando a separação de conceitos e a facilidade de composição de diferentes elementos para a otimização de um problema específico. Desenvolvimentos foram feitos principalmente numa maior separação de conceitos (incluindo a separação da dominância –`Dominance`–, que antes era estritamente a de

Pareto), no desenvolvimento de indicadores de qualidade –`Indicators`– (que não existiam), na criação de um gerador automático de experimentos de otimização –`Experiment`– (que não existia), na criação de novos problemas de otimização multiobjetivo –`MOP`– (que não existiam, o que implica a criação de novos criadores de genótipo, tradutores para fenótipo e novos avaliadores do fenótipo), e também novos genótipos –`Genotype`–, que incluem as adaptações de inspiração biológica que serão descritas a seguir, no próximo capítulo. Esse *framework* foi então integrado à suite de ferramentas da ADESD para permitir seu uso na exploração do espaço de projeto em sistemas embarcados projetados com a metodologia ADESD.

CAPÍTULO 6

UM MODELO EVOLUCIONÁRIO COM ADAPTAÇÕES BIOINSPIRADAS

Um bom cientista é uma pessoa com boas ideias originais. Um bom engenheiro é uma pessoa que faz um projeto que funciona com tão poucas ideias originais quanto possível.
—FREEMAN DYSON (Disturbing the Universe, 1979)

Não siga a estrada, apenas; ao contrário, vá por onde não haja estrada e deixe uma trilha.
—RALPH WALDO EMERSON

Este capítulo apresenta as diferentes características que foram incluídas a um modelo evolucionário para otimização multiobjetivo e é dividido em quatro seções principais. A seção 6.1 relembra os problemas sendo tratados e as adaptações com inspiração biológica que foram desenvolvidas para tratá-los. A seção 6.2 apresenta individualmente os vários aspectos de inspiração biológica, como eles foram modelados computacionalmente e incluídos ao modelo evolucionário. Também apresenta os demais aspectos que podem variar para posteriormente avaliar sua eficiência em termos de otimização multiobjetivo, sempre destacando sua motivação no contexto da exploração do espaço de projeto em sistemas embarcados. A seção 6.3 mostra como os aspectos de inspiração biológica apresentados na seção anterior podem ser usados para compor um modelo evolucionário que representa um sistema embarcado e sobre o qual pode-se fazer a exploração do espaço de projeto. Por fim, a seção 6.4 apresenta brevemente o modelo computacional de um indivíduo, focando em seu genótipo.

6.1 Introdução

Nos EA tradicionais, a estrutura básica é a de genes agrupados num cromossomo, que forma um indivíduo. De forma geral, os genes são todos

de mesmo tipo e de mesmo tamanho (fixo e conhecido), formando um cromossomo também de tamanho fixo e conhecido. Essa estrutura é adequada para muitos problemas, como a otimização de funções, em que os genes são variáveis dessas funções. Nesses casos, a ordenação linear de n genes, em que um gene g_i possui um mapeamento direto à variável x_i da função, é suficiente para representar uma solução do problema, que é basicamente um vetor de valores para as variáveis da função sendo otimizada. A posição do gene no cromossomo (seu índice no vetor) identifica qual variável ele representa, e essa posição é sempre mantida. A taxa de aplicação dos operadores de variação (tipicamente mutação e recombinação) são normalmente fixados empiricamente como parâmetros do otimizador e são idênticos para todos os indivíduos. A interação entre os genes não é considerada na estrutura, e o projetista precisa codificar genes relacionados em posições consecutivas no cromossomo, visando minimizar o efeito disruptivo do operador de recombinação, que não tem informações sobre epistasia e pode, efetivamente, separar genes relacionados nos descendentes. A representação de um possível indivíduo na estrutura clássica de EA é apresentada na figura 6.1. Nessa figura, o genótipo (*Genotype*) do indivíduo possui apenas um vetor de genes que são variáveis reais, cujos valores são os apresentados. O indivíduo representado nessa figura corresponde a uma solução da fronteira de Pareto para o problema ZDT6, um dos problemas do conjunto de problemas de teste ZDT, que oferece dificuldade aos otimizadores pois as soluções estão concentradas numa região da fronteira de Pareto.

Nesta tese retomamos a inspiração biológica que originou os algoritmos evolucionários à procura de soluções que os tornem mais diretamente aplicáveis à representação de problemas complexos que não são tão facilmente redutíveis a funções de variáveis reais ou a outros modelos matemáticos que podem ser codificados no EA como um simples vetor de variáveis reais. Esse é o caso do projeto de sistemas embarcados, especificamente nos níveis de granularidade que se deseja operar, conforme ilustrado na seção sobre o projeto dirigido pela aplicação (seção 4.1.3). Em especial, os dois principais problemas na exploração do projeto de sistemas embarcados que se deseja tratar são:

Primeiro, a representação (num sistema embarcado) de sistemas embarcados dirigidos pela aplicação e baseados em componentes. A maioria das abordagens atuais representa no modelo evolucionário apenas as variáveis que se deseja otimizar, como frequência do processador, largura do bar-

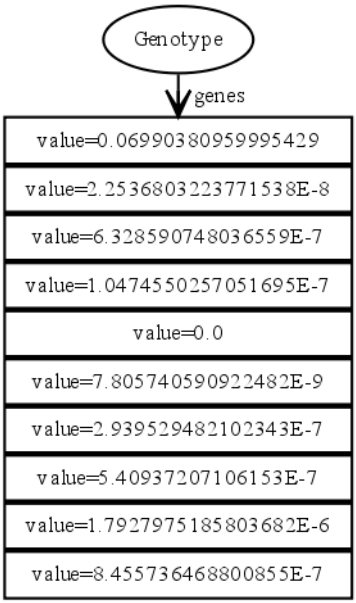


Figura 6.1: *Representação de um indivíduo num EA clássico. O indivíduo é, basicamente, um vetor (cromossomo) de variáveis reais (os genes) que representam as incógnitas de uma equação ou mais equações a ser otimizada(s).*

ramento e tamanho de cache, por exemplo, quando a exploração é feita em nível arquitetural, ou então a associação entre processos a processadores, por exemplo, quando a exploração é feita em nível de sistema. Embora funcionais, esses modelos não representam componentes, apenas variáveis dissociadas a serem otimizadas. Componentes, como definidos pela ADESD, representam entidades significativas no contexto da aplicação, são organizados em famílias, possuem configurações (*traits*), características funcionais configuráveis (*configurable features*), dependências de outros componentes ou famílias e mesmo suas características funcionais configuráveis podem conter configurações e dependências próprias (ver seção 5.3). Componentes de software podem ainda ser independentes de arquitetura ou podem ser específicos para determinada arquitetura, restringindo possíveis mapeamentos. O mesmo ocorre com componentes de hardware sintetizável que, às vezes, podem con-

ter dependências de tecnologia que permitam seu mapeamento apenas a PLDs específicos. Com isso, tanto as configurações, quanto a habilitação ou não das características funcionais (e suas configurações e dependências), as dependências alternativas de cada componente e seus mapeamentos são todos alvos de exploração e referem-se a cada componente que pode compor o sistema. Acrescente-se também que componentes alternativos mas que executam o mesmo serviço também precisam ser escolhidos exclusivamente. Portanto, busquei uma solução que permita representar diferentes características de um componente, e não apenas variáveis reais. Essas características podem ser variáveis discretas, constantes numéricas ou qualitativas e *links* para outros componentes (como arquiteturas ou PLDs), entre outros. Também é necessário que essas características refiram-se inequivocamente ao mesmo componente no modelo evolucionário. Isso significa, em princípio, que elas não podem ser divididas, pois fazem parte de uma mesma entidade básica que compõe os sistemas embarcados, que é o componente.

Segundo, a exploração em vários níveis hierárquicos, desde o nível de sistema (associando software a hardware) até o nível de arquitetura (definindo interconexões entre blocos funcionais do hardware). Não foi considerada nesta tese a exploração em níveis mais baixos, como a exploração de elementos lógicos do hardware (circuitos digitais) ou mesmo físicos (*place and routing* de transistores, *layout* físico de chips). Assim, considero que um sistema embarcado é uma composição de componentes pré-desenvolvidos (software / IPs) e pré-manufaturados (hardware físico), mas a manufatura de um novo chip (ASIC) não faz parte da exploração. Como foi apresentado na seção 4.2.4, existem várias abordagens para a exploração hierárquica. Contudo, busquei representar as informações para exploração em todos esses níveis no mesmo modelo evolucionário, de forma a ter um modelo único e poder escolher, possivelmente "em tempo de exploração", quais níveis e características explorar, conforme restrições de projeto e do projetista, como o próprio tempo disponível para exploração. Desse modo, buscou-se uma solução de inspiração biológica que permita manter partes da solução inalteradas (sem exploração) enquanto outras evoluem, e que esse comportamento possa ser alterado automaticamente durante a otimização, possivelmente pela própria passagem do tempo físico (tempo de otimização), ou pela diversidade da população (liberando outras partes do sistema para exploração após a estagnação da população), por exemplo.

Para tratar os problemas mencionados procurei alterar os modelos evo-

lucionários com soluções inspiradas na própria genética. As alterações realizadas nesta tese limitam-se a mudanças na estrutura do indivíduo que representa uma solução ao problema sendo otimizado e, consequentemente, aos operadores genéticos que modificam essa estrutura. Contudo, a estrutura usada para representar as soluções é talvez a característica mais importante de um EA [Bonissone e Subbu 2007]. Não foram alterados outros componentes dos algoritmos evolucionários, como estratégias de seleção e os princípios gerais de funcionamento (ver seção 3.2.1). Assim, incorporei à estrutura do indivíduo os seguintes elementos:

1. A auto-adaptação de parâmetros dos operadores genéticos de forma independente para cada gene e mesmo para cada códon, representando o efeito de elementos epigenéticos e da pressão ambiental sobre elementos genéticos mais ou menos importantes, já que nos organismos reais a taxa de mutação é muito diferente entre genes ou mesmo entre códons do mesmo gene, de forma a manter estáveis os elementos com propriedades funcionais importantes (ou seja, que participam de sítios ativos). A auto-adaptação dos parâmetros permite retirar do projetista a responsabilidade de determinar esses parâmetros, de avaliar sobre cada gene ou códon a pressão de seleção e de identificar indiretamente as variáveis mais importantes num problema (pela pressão de seleção sofrida). A possibilidade de ter taxas de aplicação dos operadores genéticos diferenciadas para cada gene ou códon permite manter alguns elementos da solução constantes, enquanto outros podem ser explorados. Isso pode auxiliar no tratamento do problema da exploração em um ou mais níveis no contexto de sistemas embarcados;

2. A neutralidade genética, representada pelo controle da expressão genética, pelo multiploidismo, por regiões intragênicas e por genes não codificantes. Nos organismos reais, uma parte considerável do genótipo não é expressa nunca (lixo genético), e a maior parte do genótipo não é expressa num dado momento. Toda a redundância e formas de neutralidade podem ser proveitosas na manutenção para gerações futuras de genes que já foram úteis no passado, além de sua importância vital no processo embrionário e de desenvolvimento dos organismos, bem como de sua especialização celular, casos que também costumam ser ignorados pelos modelos computacionais atuais. No contexto de sistemas embarcados, a neutralidade genética permite que

haja informações de controle no modelo da solução e que não são utilizadas na geração do sistema final (tradução para fenótipo); também permite que componentes alternativos de uma solução (como uma versão otimizada para área e outra para desempenho, por exemplo) compitam entre si como genes alelos para serem expressos no sistema gerado, ou mesmo que componentes que formaram boas soluções no passado sejam mantidos em gerações futuras, mesmo que neutros, de modo que possam ser mais facilmente reutilizados se ainda estiverem no genótipo do que se tiverem que ser gerados novamente por mutação;

3. Mudanças epigenéticas, como permutação, silenciamento de genes e regulação da expressão genética, representando o ambiente celular no qual insere-se o material genético, e cuja pressão de seleção é imediata aos genes, quando considerados os elementos básicos da evolução. Pesquisas tem, cada vez mais, descoberto a influência do ambiente bioquímico intracelular na genética e que representam, até certo ponto, mudanças genéticas causadas por fatores fenotípicos. No contexto do projeto de sistemas embarcados, a epigenética pode auxiliar no tratamento dos problemas mencionados. Em relação à exploração multihierárquica, taxas de mutação de certos elementos da solução (que se deseja explorar ou não) podem ser alteradas conforme informações não-genéticas, como a diversidade da população, intensidade de seleção ou mesmo o tempo decorrido na exploração. Informações não-genéticas podem também definir o comportamento da recombinação, impedindo, por exemplo, que elementos correlacionados sejam separados em gerações futuras, o que permite o tratamento da epistasia e também manter certas composições de componentes, como o suporte de hardware ou sua interconexão, por exemplo.

6.2 Modelo Evolucionário Bioinspirado

As características de inspiração biológica foram modeladas de forma que cada uma delas possa ser incluída ou não no modelo evolucionário, independentemente das demais características, conforme especificação do projetista. Assim, cada característica será descrita separadamente, incluindo sua possível utilidade na representação de problemas complexos e um exemplo

da estrutura utilizada para representá-la. Iniciamos com as características genéticas e depois as epigenéticas, das mais gerais às mais específicas.

6.2.1 Multiploidismo

Nos organismos mais evoluídos, o material genético está organizado em unidades isoladas, os cromossomos. Modelos evolucionários multi-cromossômicos são relativamente comuns e são geralmente usados para representação de sub-problemas. Entre eles pode-se citar a própria GEP (GEP - *Gene Expression Programming*), em que cada cromossomo corresponde a um termo que será posteriormente unido aos demais (através de operadores aritméticos) para formar a expressão completa de uma equação que representa a solução. Contudo, se todos os cromossomo são agrupados linearmente para formar o genótipo, e cada cromossomo é uma estrutura linear em que todos os genes são expressos, então um indivíduo multi-cromossômico pouco ou nada difere de um indivíduo com um único e grande cromossomo. Desse modo, considero essa representações simplistas.

Em organismos organizados em cromossomos, genes com funções correlacionadas, ou que cooperam entre si, costumam estar no mesmo cromossomo e ainda estar fisicamente próximos entre si, de modo que é minimizada a probabilidade de serem separados por recombinação na próxima geração. Assim, nos organismos vivos, a organização em cromossomos auxilia o tratamento da epistasia. Entre outra de suas várias utilidades, destaca-se uma que ocorre nos organismos diploides, no qual há sempre um par de cada cromossomo (homólogos). Nesses organismos, os genes em posições correspondentes (*loci*) em cada cromossomo de um par competem para serem transcritos e, assim, influenciar no fenótipo do indivíduo. Isso significa que duas características costumam ser ignoradas nos modelos evolucionários usuais: (1) A neutralidade genética decorrente de um dos genes alelos não ser transcrito por ser recessivo em relação ao seu par; e (2) Genes competem entre si como fazem os indivíduos. De fato, parecem ser os genes, e não os indivíduos, as entidades básicas sobre o qual atua a evolução [Dawkins 1976]. Isso significa que o ambiente bioquímico no qual estão inseridos os genes (e no qual os genes sofrem seleção) pode ter grande importância e sua representação não deveria ser ignorada nos modelos computacionais.

Nesta tese, o diploidismo é representado de maneira genérica como

"multiploidismo", permitindo que duas ou mais versões diferentes de genes alelos coexistam no mesmo *locus* de cromossomos homólogos. Entretanto, ao invés de representar muitos cromossomos que formam pares (ou conjuntos maiores, nos casos de triploidismo, por exemplo), o modelo adotado apresenta cada *locus* de cada cromossomo como uma lista de genes. A figura 6.2 apresenta a estrutura de um cromossomo multiploide. A quantidade de genes em cada *locus* é fixada num número específico, como 2 (dois), caso em que se representam organismos diploides (como os humanos), ou mesmo 1 (um), caso em que se representa a estrutura clássica de um cromossomo como um vetor de genes. É importante lembrar que um cromossomo modelado como um conjunto de listas para 2 (dois) genes representa, na realidade, dois cromossomos homólogos diferentes num organismo diploide. Preferiu-se organizar os genes por locus e não por cromossomos pois os operadores de variação irão operar sobre genes alelos e, com isso, essa organização torna-se mais eficiente computacionalmente. Assim, na figura 6.2, o *gene l.c* representa o gene do *l-ésimo locus* do *c-ésimo* cromossomo. Os três cromossomos apresentados nesta figura são destacados pelas mesmas cores em seus genes.

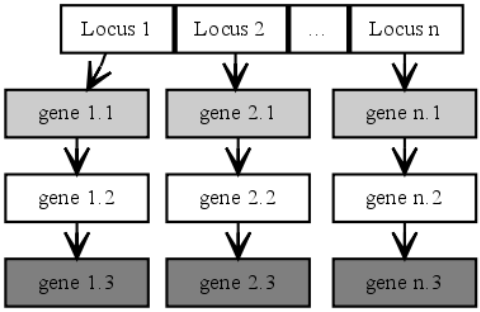


Figura 6.2: *Cromossomo multiploide. Cada locus do cromossomo é uma lista de genes alelos. A quantidade de genes em cada lista pode ser fixado num inteiro qualquer, como 2, formando indivíduos diploides, ou mesmo 1, retornando ao modelo clássico em que o cromossomo é apenas um vetor de genes. A figura ilustra um indivíduo triploide.*

A estrutura de multiploidismo reflete-se nos operadores genéticos de variação do seguinte modo: o operador de recombinação tem sua funcionalidade alterada, pois a estrutura de um cromossomo, na realidade, representa

uma quantidade de cromossomos que corresponde à quantidade de genes nas listas. Portanto, o operador precisa ser aplicado a esse cromossomo essa mesma quantidade de vezes (uma para cada índice nas listas). Os operadores de adição, remoção e cópia precisam operar sobre *loci*, e não sobre genes, ou seja, operar sobre a quantidade de genes em todos os cromossomos homólogos, e não apenas sobre um.

6.2.2 Competição Genética

Outras características biológicas derivadas do multiploidismo incluem a competição de genes pela transcrição e, como consequência, a possível neutralidade de alguns genes (os recessivos). Assim, quando há mais de um gene no mesmo *locus*, apenas um deles ou então mais de um podem participar do fenótipo do indivíduo, dependendo se um gene é dominante, recessivo ou codominante em relação aos seus alelos. Essas características foram representadas por: (1) Uma estrutura que precede cada gene e que contém informações diversas sobre esse gene (a região promotora), incluindo sua dominância; e (2) O mecanismo de tradução do genótipo em fenótipo passa a avaliar as informações de dominância em genes alelos para definir como eles serão traduzidos. A figura 6.3 apresenta três genes alelos com a estrutura que contém a informação sobre suas dominâncias.

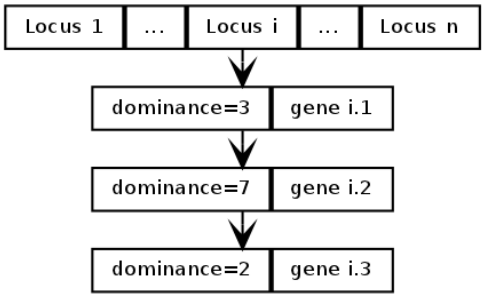


Figura 6.3: *Genes alelos e dominância. Cada gene alelo de um locus possui uma estrutura (epigenética) que contém a informação sobre sua dominância e que permite a competição entre esses genes e diferentes manifestações no fenótipo do indivíduo.*

A dominância de um gene pode ser definida por muitas maneiras di-

ferentes. Nesta tese, a dominância de um gene é aumentada ou diminuída durante a etapa de seleção, proporcionalmente à quantidade de outros indivíduos da população que são dominados (no sentido de soluções preferíveis, como dominância de Pareto) pelo indivíduo que contém esse gene. Essa abordagem introduz um efeito epigenético no modelo, uma vez que características fenotípicas (a aptidão do indivíduo) altera sua informação genética. A informação de dominância é utilizada apenas pelo tradutor de genótipo em fenótipo, do seguinte modo: inicialmente é calculada a dominância relativa de cada gene alelo, dada pela equação (6.1), onde $d_{l,i}$ é a dominância do *i-ésimo* alelo do *locus* l, u é a quantidade de genes alelos por *locus*, ou seja, a quantidade de cromossomos homólogos (dois, no caso de diploides, por exemplo), e $\widehat{d}_{l,i} \in \Re[0,1]$ é a dominância relativa desse alelo.

$$\widehat{d}_{l,i} = \frac{d_{l,i}}{\sum_{j=1}^{u} d_{l,j}} \tag{6.1}$$

Se um gene g_{l_i} tiver dominância relativa $\widehat{d}_{l,i} > 0.5$ ele é considerado dominante e os demais recessivos, e apenas o valor do gene dominante será considerado na definição do fenótipo. Caso contrário, ou seja, todos os genes têm dominância relativa $\widehat{d}_{l,i} \leq 0.5$, então eles são considerados codominantes, e os valores de todos eles podem ser considerados no fenótipo, proporcionalmente à sua dominância relativa. No caso de genes que codificam variáveis reais, isso significa que o valor do *locus* (v_l) corresponde a uma média ponderada (pela dominância relativa) dos valores de todos os genes alelos desse *locus*, ou seja, $\mathfrak{m}(\overrightarrow{g_l}) \mapsto v_l$, $\mathfrak{m}(\overrightarrow{g_l}) = \mathfrak{m}(g_{l,1}, g_{l,2}, ..., g_{l,u}) = \sum_{j=1}^{u} \widehat{d}_{l,j}.v_{l,j}$

Porém, como também proponho nesta tese, os genes podem representar outros tipos de variáveis, como variáveis discretas, qualitativas ou mesmo equações envolvendo outras variáveis ou então "links" ou "ponteiros" para outros genes. Cada tipo de variável precisa definir (implementar) uma forma específica de "proporcionalidade" para gerar o valor correspondente da codominância de vários genes desse tipo. Se codominância não fizer sentido a um tipo de variável, como um "link", por exemplo, uma implementação simples poderia considerar apenas o valor do gene com maior dominância, mesmo que não seja realmente dominante ($\widehat{d}_{l,i} > 0.5$).

No contexto dos sistemas embarcados, variáveis discretas podem re-

presentar configurações como largura de barramentos (8, 16, 32, 64 bits) ou taxas de transmissão (9600, 14400, 19200, 28800 bps, etc). Variáveis qualitativas podem representar parâmetros de compilação (*-Os, -Ofast, -fno-default-inline, -funroll-loops*, etc) ou de síntese (*-ol std, -pl med, -rl high*, etc) que regulam o esforço dessas ferramentas na minimização de código ou de tempo de execução, por exemplo. Genes que representam equações podem ser usados para a evolução de modelos de estimação para expressões desconhecidas de funções-objetivo, substituindo outros modelos analíticos, como modelos de regressões ou de lógica difusa, por exemplo. Também podem representar características relacionadas num componente (como a relação tensão de entrada / tensão de saída num regulador de tensão, ou então V_{CE} em relação a V_{BE} e β num transistor, por exemplo). Por fim, genes que representam "links" ou "ponteiros" para outros genes (ou outras estruturas genéticas, como cromossomos) podem ser usados para realizar o mapeamento de componentes de software a arquiteturas (como a alocação de processos a processadores), de componentes de hardware sintetizável a PLDs (associando *IPs cores* às FPGAs em que serão sintetizados), ou então variáveis que devem ter seus valores iguais ou relacionados (como todos os barramentos com a mesma largura, por exemplo)[1].

Duas abordagens principais e alternativas poderiam ser utilizadas para representação mais abrangente das variáveis que podem ser codificadas geneticamente: (1) Utilizar uma codificação uniforme e relativamente simples para todos os tipos de variáveis, como a *codificação binária* (ver seção 3.3.1) ou outra forma discreta (usando apenas símbolos *A, T, C*, e *G*, por exemplo), o que fornece grande flexbilidade e facilidade na codificação, mas traz problemas na decodificação e tradução do genótipo para o fenótipo; ou (2) Utilizar formas diferentes de codificação para diferentes tipos de variáveis, como a *codificação por variáveis reais* (para variáveis reais ou mesmo inteiras) ou adaptações da codificação utilizada na GEP para variáveis discretas, qualitativas e outras, o que "amarra" a codificação a uma variável específica, mas facilita a decodificação e tradução do genótipo para fenótipo.

[1] Quando mais de uma variável têm sempre exatamente o mesmo valor (ex. $V_1 = V_2$), ou uma é função de outra (ex. $V_1 = 3.V_2/5$), pode-se considerar que se tem, de fato, uma única variável, o que é verdade. Porém, em casos práticos, pode ser mais fácil expressar a relação entre as variáveis e incluí-las (automaticamente) no modelo do que incluir apenas uma e depois inferir as outras. O primeiro caso facilita a tradução do genótipo para o fenótipo, enquanto o segundo facilita o criador do genótipo.

Ambas alternativas possuem pontos interessantes, mas optou-se pela segunda abordagem nesta tese por sua facilidade em evitar e tratar codificações e soluções inválidas, além de facilitar a tradução para o fenótipo, que representa uma solução ao problema, e que no caso dos sistemas embarcados poderia ser muito complexo de ser obtido a partir de codificações mais elementares. Contudo, proponho ainda que as variáveis associadas ao problema não sejam representadas nos genes, mas em estruturas genéticas mais elementares.

6.2.3 Códons

Nos organismos vivos, genes são compostos por estruturas mais elementares de informação: os códons, que codificam a informação necessária para a síntese de um aminoácido, e que são compostos por três bases nucleicas. Um gene é, então, um conjunto de códons que, juntos, especificam um elemento único e completo, que é uma proteína. Numa analogia com os sistemas embarcados, informações elementares como taxa de transmissão, quantidade de bits, tipo de paridade e realização ou não de cálculo de CRC, juntas, podem especificar um elemento único e completo, como um USART (um componente). Representar cada uma dessas variáveis como genes não apenas "desestrutura" os componentes, como causa problemas de epistasia, pois todas essas informações são altamente inter-relacionadas, e também impede o uso de certos operadores genéticos de variação, como a transposição, por exemplo, pois a alteração de posição no cromossomo alteraria a semântica da informação. Poderia-se propor a representação de tais informações como genes, considerando os cromossomos como componentes, ou seja, não haveria necessidade de incluir códons no modelo; porém, os genes deveriam ser independentes (e não o seriam nesse caso) e a inspiração biológica de organismos com múltiplos cromossomos pareceria distorcida, pois a recombinação de cromossomos agravaria a epistasia e não geraria a variação genética a que se propõe (pois apenas informações de um componente, e não conjuntos de vários componentes, seriam variados).

Assim, introduzi o conceito biológico de códon na estrutura do modelo evolucionário, o que não existe em outras soluções. De maneira geral, o genótipo de indivíduo é composto por cromossomos; cada cromossomo é composto por *loci*; cada *locus* é composto por genes; e cada gene é composto

por códons, que são a unidade básica de informação genética[2]. A figura 6.4 apresenta essa estrutura. Como o operador de recombinação opera sobre genes, e não sobre códons, então todos os códons (que representam variáveis inter-relacionadas) serão transmitidos ao mesmo descendente, tratando a epistasia. Contudo, o operador de mutação precisa ser modificado de duas formas: (1) Precisa operar diretamente sobre os códons, e não mais sobre os genes; e (2) Precisa ser específico para cada tipo de códon, ou seja, cada tipo de códon deve implementar sua própria versão do operador de mutação que opera sobre ele. Um operador de mutação "genérico" varre o genótipo do indivíduo e, para cada códon, invoca uma especialização do operador para aquele tipo de códon.

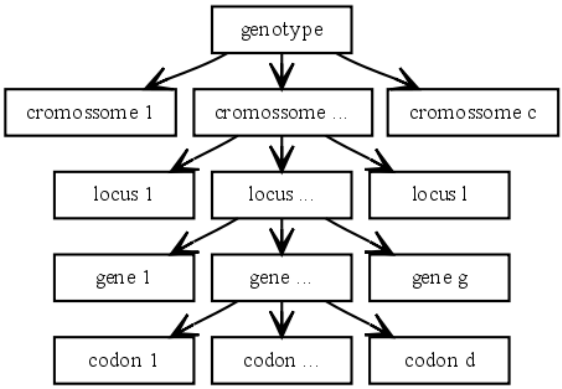


Figura 6.4: *Estrutura geral do genótipo de um indivíduo. Um genótipo pode ser constituído de vários cromossomos independentes. Cada cromossomo (ou cromossomo-par) possui um conjunto de loci, sendo que cada locus pode ser ocupado por um gene ou vários genes alelos. Por fim, cada gene pode ser composto de vários códons, que representam variáveis elementares.*

Os principais tipos de códons desenvolvidos nesta tese incluem: (1) Variáveis reais; (2) Expressões matemáticas; e (3) Variáveis discretas, qualitativas ou links. Variáveis reais são amplamente utilizadas e sua mode-

[2]Em termos biológicos, pode-se argumentar que é o nucleotídeo, e não o códon, a unidade básica de informação. Contudo, um nucleotídeo em si não representa ou codifica uma informação completa. Apenas um conjunto de três nucleotídeos, ou códon, é que possui um significado definido, que é o de representar um aminoácido específico a ser sintetizado.

lagem é imediata, possivelmente com definição de limites inferior e superior, conforme especificação do projetista. A figura 6.5 apresenta a estrutura de um gene que é composto por três códons que representam variáveis reais com limites superior e inferior. Quanto especifica-se limites para os valores da variável, qualquer valor inválido (fora dos limites) gerado por uma mutação é automaticamente corrigido, deslocando-o para dentro da faixa permitida. Como foi mencionado anteriormente, cada tipo de códon precisa especificar o funcionamento do seu operador de mutação. Algumas variantes para o operador de mutação foram desenvolvidas, dentre as quais uma abordagem comum e apresentada na equação 3.8 (página 64). Nessa abordagem, o valor do códon é adicionado a um valor aleatório que segue uma distribuição de probabilidade especificada. Ajustando a distribuição e/ou seus parâmetros, é possível controlar a "força" da mutação.

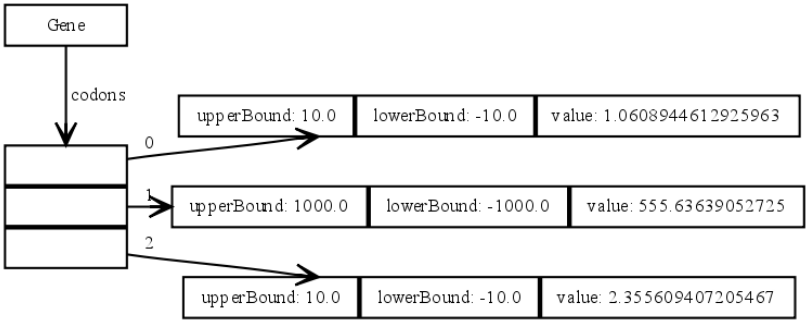


Figura 6.5: *Exemplo da estrutura de códons de variáveis reais. Este gene é constituído por três códons que representam variáveis reais com limites superior (upperBound) e inferior (lowerBoud).*

Expressões matemáticas são modeladas conforme sugerido pela GEP (ver página 56), ou seja, os códons (e não mais os genes) possuem um cabeçalho (com operadores e terminais), uma cauda (apenas com terminais) e constantes. A figura 6.7 apresenta o exemplo de um gene que contém um único códon que representa uma expressão matemática. Nessa figura, o conjunto de operadores é $\{+,-,*,/\}$, o conjunto de terminais (as variáveis da equação) é $\{x,y\}$, o conjunto de constantes é $\{530.0, 631.8, 687.0, -908.0\}$ e o conjunto de seus índices é $\{1,2,3,0\}$. O valor do códon (cabeçalho e cauda) apresentado na figura 6.7 é "$+-*??xy$" (ver *links* em *head* e *tail*),

de modo que a expressão representada por esse códon é $cte_1 - cte_2 + x.y = 631.8 - (-687) + x.y$, pois ao valor do códon corresponde a árvore de expressão da figura 6.6, e os dois primeiros índices para constantes são 1 e 2.

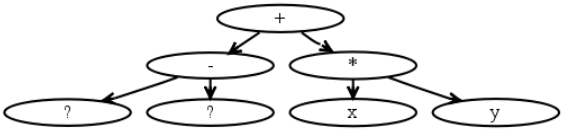


Figura 6.6: *Árvore de expressão do códon GEP. A string "+-*??xy" corresponde à árvore de expressão apresentada, em que o símbolo "?" representa uma das constantes do códon, conforme seu conjunto de índices.*

Variáveis discretas, qualitativas ou *links* foram modeladas como variações do gene GEP. Nesses casos, fez-se $h = 1$, $\mathscr{F} = \{\}$ e $\mathscr{T} = \{?\}$ (ver página 56), ou seja, o cabeçalho foi limitado a um caractere, que deve ser um operador ou um terminal; porém, o conjunto de operadores $\mathscr{F}$ é vazio, de modo que o cabeçalho deve ser um terminal, e o único terminal definido é o símbolo de constantes "?". Portanto, a expressão é reduzida a uma única constante da lista de constantes e, mais especificamente, àquela cujo índice for o primeiro da lista de índices de constantes. A única diferença entre variáveis discretas, qualitativas ou *links* é que nas primeiras as constantes são inicializadas com os possíveis valores discretos numéricos, a segunda são inicializadas com os possíveis valores alfanuméricos categóricos (qualitativos) e na última são inicializadas com ponteiros para os objetos-alvo. A figura 6.8 apresenta um exemplo de estrutura de um gene com um único códon que representa uma variável discreta. No exemplo apresentado, a variável possui sete possíveis valores discretos e que correspondem a taxas de transmissão de uma USART. Variáveis tipo "link" são úteis, por exemplo, para representar dependências alternativas entre componentes (ver exemplo 4.1 e figura 4.4 na página 80). Nesses casos, as constantes são inicializadas com ponteiros para os genes que representam os componentes alternativos dos quais há dependência. Uma mutação nesse códon, então, causa a mudança da dependência e a exploração de novos componentes alternativos.

Por fim, o modelo desenvolvido nesta tese permite que códons de diferentes tipos coexistam num mesmo gene. Essa característica é importante, pois as variáveis inter-relacionadas, representadas como códons num mesmo

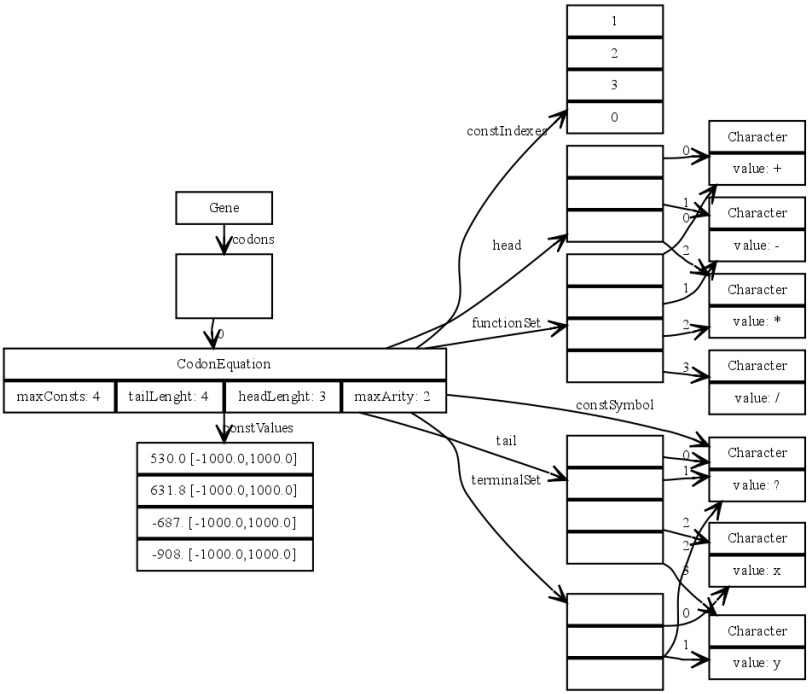


Figura 6.7: *Exemplo da estrutura de códon de equação. Este gene é constituído de um único códon GEP que representa uma equação matemática.*

gene, nem sempre são do mesmo tipo. A existência de genes heterogêneos não afeta o funcionamento dos operadores de variação que operam sobre cromossomos e genes (como recombinação, transposição, adição e remoção), e o operador de mutação é especificado para cada diferente tipo de códon, de forma que para cada códon invoca-se uma especialização do operador para aquele tipo de códon, independentemente dos demais códons serem do mesmo tipo ou não.

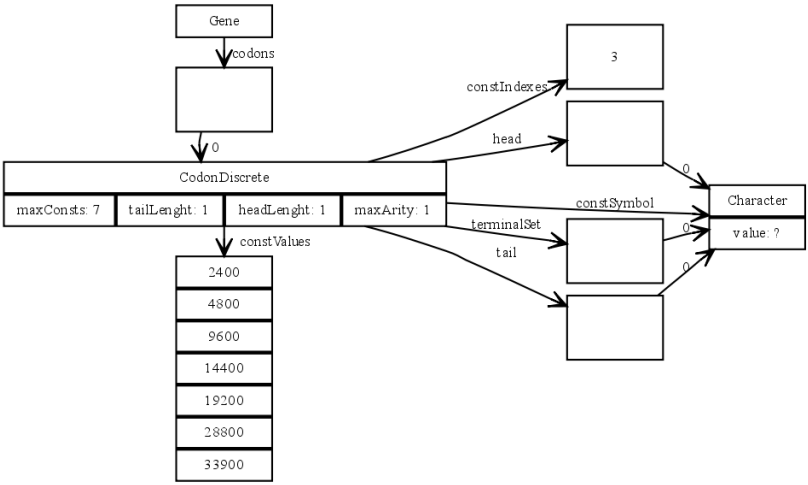


Figura 6.8: *Exemplo da estrutura de códon de variáveis discretas. Este gene é constituído de um único códon de variáveis discretas, uma adaptação do GEP em que as constantes são os valores possíveis da variável e a equação restringe-se a uma única constante. No exemplo apresentado a variável representa possíveis taxas de transmissão numa USART.*

6.2.4 Região Promotora

Os desenvolvimentos apresentados até este ponto permitem, basicamente, a flexibilização da estrutura que representa soluções do problema sendo tratado, e inclui elementos de inspiração biológica que não haviam sido representados nos demais trabalhos encontrados na literatura, ou haviam sido representados de maneira simplificada (se comparados à representação adotada aqui). Em especial, destacou-se a representação hierárquica, o multiploidismo, e genes como agregados de códons, possivelmente heterogêneos. Entretanto, algumas caraterísticas de inspiração biológica de interesse nesta tese transpõem estruturas específicas (cromossomos, locus, genes, códons) e afetam todas ou apenas algumas delas, e outras sequer referem-se diretamente à informação genética, mas modificam o modo pelo qual a evolução opera. O elemento que passa a ser descrito agora permite adaptar operadores genéticos para cada elemento da estrutura do indivíduo (cromossomo, locus, gene, có-

don), permite neutralidade genética, competição por transcrição, o tratamento de soluções inviáveis e auto-correção da estrutura, e evolução de parâmetros genéticos; Permite avaliar a vantagens de gerar ciclos de grande pressão de seleção, de mudar a taxa de mutação conforme a diversidade da população, de fazer exploração hierárquica e criar *templates*, entre outras ações, e foi denominado de *Região Promotora*.

Em termos biológicos, a *Região Promotora*, ou *Sítio Promotor*, é um trecho de DNA que antecede os genes (geralmente por poucas dezenas de bases nucleicas) e que informa o ponto de início de transcrição e influencia na regulação da expressão daquele gene (ver seção 3.1). Nesta tese, o modelo computacional dessa estrutura desempenha essas e ainda outras funções que são realizadas (em organismos reais) por diferentes estruturas genéticas ou epigenéticas. A decisão de concentrar diferentes funções evolutivas nessa única estrutura auxilia em seu reuso nos diferentes elementos genéticos (cromossomo, locus, genes, códons) e no desenvolvimento de novos operadores genéticos, e a decisão de manter sua denominação de Região Promotora deve-se ao fato de seu modelo computacional poder preceder cada um dos diferentes elementos genéticos, regendo seu comportamento.

6.3 Modelo Evolucionário para Exploração de Espaço de Projeto

O Modelo Evolucionário para Exploração do Espaço de Projeto, ou M3EP, é basicamente a aplicação do modelo evolucionário com as adaptações apresentadas na seção anterior no contexto do projeto de um sistema embarcado dirigido pela aplicação. Ele possui uma nova estrutura evolucionária que inclui elementos do processo bioquímico e genético e que são normalmente ignorados nas abordagens atuais e adaptações também nos operadores genéticos que a manipula, como foi previamente apresentado. Contudo, ele agrega também as "saídas" do projeto de um sistema embarcado (conforme o metamodelo de sistema embarcado dirigido pela aplicação, apresentado no capítulo 5, seção 5.1), que incluem os componentes que são especificados no metamodelo de componentes embarcados (seção 5.3), distinguindo o software, hardware sintetizável e hardware físico para reuso, e também é parte do contexto no qual estimativas para os custos do sistema (caracterizações

passíveis de exploração) podem ser obtidas. Desse modo, ele é um agregador dos modelos desenvolvidos. O M3EP deve ser capaz de representar:

- Uma ou mais instâncias de cada componente de software, de hardware sintetizável e de hardware físico. A unidade de reusabilidade proposta pela ADESD para componentes de software é a classe, e é o código da classe que é incluído (compilado e ligado) na imagem final do sistema. Também é a classe que é adaptada por aspectos, funcionalidades configuráveis e parâmetros, que valem para todas as instâncias da classe, de modo que, para cada imagem de software, uma única instância de cada componente de software é necessária. Entretanto, cada imagem de software é mapeada a um elemento processador, e arquiteturas multiprocessadas exigem mais de uma instância de cada componente. A necessidade de mais de uma instância para componentes de hardware é mais clara. Fica explícito, todavia, que o modelo pode ter uma quantidade variável de genes, o que não é comumente tratado em outras classes de EA.

- As configurações (*configurable features*), parametrizações (*traits*) e adaptações (*aspects*) de cada instância de componente previstas na ADESD. Mais do que uma lista de valores reais a serem codificadas como genes no cromossomo, esses elementos exigem diferentes tipos de variáveis e uma estrutura hierárquica, já que configurações podem ter seus próprios parâmetros (e dependências). Dado que componentes de software, hardware sintetizável e físico têm diferentes características, parâmetros e diferentes configurações (também com diferentes parâmetros), fica claro que genes têm estruturas e tamanhos diferentes, o que não é tratado em outras classes de EA.

- Diferentes formas de dependências entre componentes, que podem ser: *Requisit*, *Exclude*, e *Alternative requisit* (seção 5.3). Componentes formam um grafo dirigido de dependências entre componentes, e para dependências do tipo "*requisit*" todos os componentes recursivamente dependentes (nodos alcançáveis no grafo de dependências) devem ser incluídos para formar uma solução viável. Para dependências do tipo "*exclude*" os componentes especificados na dependência devem ser excluídos do sistema para formar uma solução viável. E para dependências do tipo "*alternative requisit*", um e apenas um dos componentes es-

pecificados na dependência deve ser incluído para formar uma solução viável.

- Mapeamentos entre componentes de software e elementos de hardware. Esse mapeamento pode representar diferentes associações em diferentes níveis de abstração da exploração do espaço de projeto. Normalmente refere-se ao mapeamento entre tarefas (processos, *threads*) e processadores, ou entre operações elementares de software (somas, multiplicações, etc) e elementos básicos de hardware (somadores, multiplicadores, etc). Na ADESD, sendo a classe a entidade de reuso (de software), o mapeamento se dá entre componentes de software (*threads*, semáforos, alarmes, escalonadores, etc) e arquiteturas (processadores, microcontroladores, *softcores*). Mais especificamente, componentes de software são mapeados a arquiteturas, pois é necessário conhecer o processador para realizar a tradução do software para instruções de máquina da arquitetura em questão[3]. Certamente, classes podem ter diferentes granularidades, o que provê maior liberdade a esse mapeamento. Entretanto, ainda seguindo a ADESD, sugere-se sempre que elas representem entidades relevantes do domínio da aplicação.

- Mapeamentos entre componentes de hardware sintetizável e hardware físico. Todo componente de hardware sintetizável passa pelo processo de síntese de hardware e gera uma "imagem de hardware" (*bitstream*), análoga à imagem de software. De modo também análogo ao software, que deve ser armazenado numa memória associada à arquitetura no qual executará, o hardware sintetizável deve ser armazenado numa memória (Flash) associada a um PLD que terá sua configuração interna de hardware definida por esse *bitstream*. No caso de um ASIC, posteriormente deve-se ainda realizar o processo industrial de concepção do circuito, que foge ao escopo do projeto considerado nesta tese.

- Conexões físicas e lógicas entre componentes de hardware físico e sintetizável que definem, em conjunto, um suporte de hardware. Cada pino de cada componente pode (ou deve) ser conectado a outros pinos de outros componentes, formando um diagrama esquemático do circuito eletrônico do suporte de hardware.

[3]Efetivamente, o software fica armazenado numa memória que é associada ao processador.

Por representar uma solução completa de sistema embarcado, da conexão física dos componentes eletrônicos do hardware ao mapeamento de componentes de software a arquiteturas, diferentes níveis de abstração estão presentes no modelo. A figura 6.9 mostra a estrutura geral do genótipo do indivíduo e destaca a existência de cromossomos específicos para a aplicação, componentes de software, componentes sintetizáveis e componentes físicos, além de cromossomos para a interconexão de hardware sintetizável e de hardware físico. A separação desses cromossomos é essencial para a exploração e para a manutenção de algumas características citadas. Não é necessário que todos esses cromossomos estejam presentes numa solução. Por exemplo, se uma solução não incluir dispositivos lógicos programáveis, então não existirão cromossomos de componentes sintetizáveis e de sua interconexão, por exemplo. Por outro lado, se uma solução consistir de dois nodos, então alguns cromossomos serão duplicados. A figura 6.10 apresenta de forma um pouco mais detalhada (apresentando os genes/componentes de cada cromossomo) a estrutura de uma possível solução representada pelo M3EP. Nessa figura, supõe-se que a aplicação embarcada é composta por 3 *threads*: *main, T1* e *T2*, que especificam interfaces para os componentes de software `Thread` (*main*), `Semaphore` e `Alarm` (*T1* e *T2*). Esses componentes de software são mapeados a um *softcore* (`Plasma_IP`) que é sintetizado juntamente com outros componentes de hardware sintetizável (`Timer_IP` e `IC_IP`) numa FPGA Spartan3 (`XC3S250`), conectada a outros componentes físicos (`K9F5608U0C` e `Conn32p`) para formar o suporte de hardware físico.

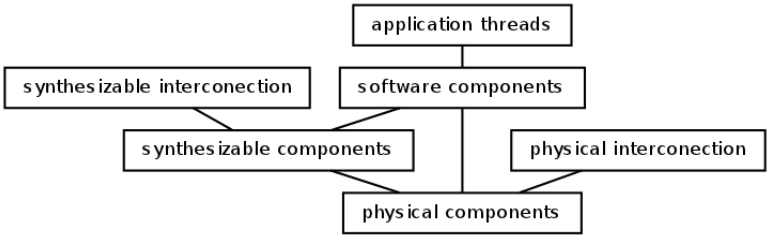


Figura 6.9: *Estrutura geral do M3EP. Um indivíduo é composto por diferentes cromossomos, que representam a aplicação embarcada, componentes de software de suporte, componentes de hardware sintetizável e sua interconexão, e componentes físicos e sua interconexão.*

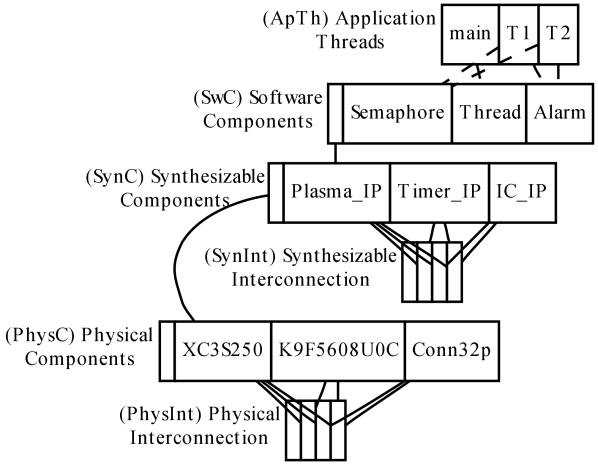


Figura 6.10: *Exemplo de estrutura de cromossomos do M3EP. Cada cromossomo possui uma região promotora e genes que representam os elementos exploráveis de cada componentes ou de sua conexão. Nesse exemplo há uma aplicação com três threads mapeadas ao softcore Plama_IP que será sintetizado no PLD XC3S250. São distintos os suportes de hardware sintetizável e físico.*

Iniciando a descrição do modelo pela aplicação embarcada, existe um cromossomo para representar cada aplicação, em que cada gene representa uma *thread*. A região promotora do gene especifica qual é a *thread* e seus códons contêm dependências alternativas para componentes de software que satisfazem seus requisitos, ou seja, que implementam as interfaces infladas especificadas nessa *thread*. Mesmo que exista um único componente que satisfaz determinada interface, essa dependência é sempre representada como uma dependência alternativa, pois o cromossomo no qual o componente está inserido define a arquitetura em que executará; portanto, se houver mais de uma arquitetura no suporte de hardware, esse mesmo componente estará presente em dois cromossomos diferentes (cada qual mapeado a uma arquitetura diferente), de modo que a dependência alternativa é também usada para o mapeamento de *threads* da aplicação a arquiteturas. A figura 6.11(a) mostra o caso em que os mesmos componentes de software (`Semaphore`, `Thread` e `Alarm`) aparecem em cromossomos distintos, um mapeado a um *softcore*

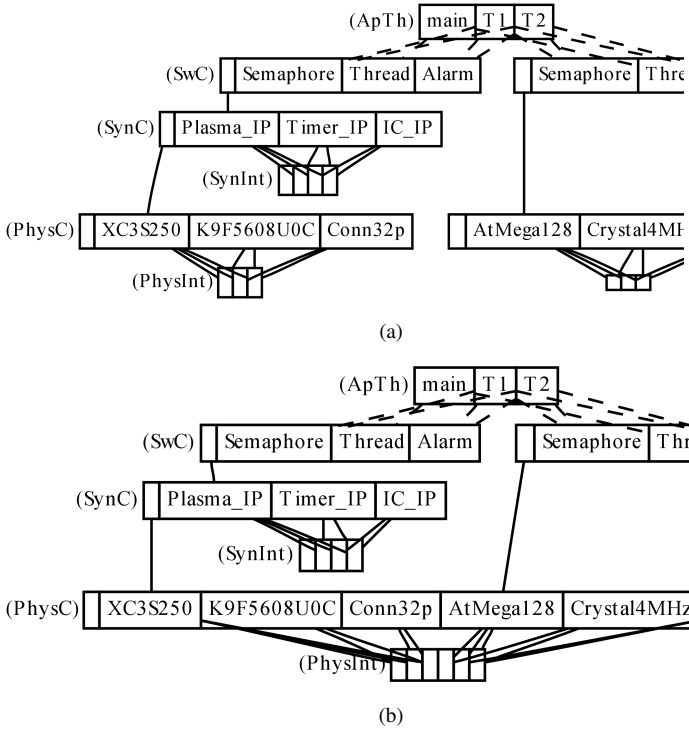


Figura 6.11: *Exemplo de estrutura de cromossomos do M3EP. (a) M3EP possivelmente com dois nodos, um deles com um PLD em que o software executa num software e outro microcontrolado, em que o software executa diretamente no microcontrolador; e (b) M3EP com um nodo multiprocessado, com uma única plataforma composta por um PLD e um microcontrolador, onde algumas threads executam no softcore e outras no microcontrolador (o mapeamento final não é especificado).*

(`Plasma_IP`) e outro mapeado a uma arquitetura física (`AtMega128`), que estão em nodos (ou suportes de hardware físico) distintos. Dependendo de como a exploração do espaço de projeto resolver essas dependências alternativas, diferentes mapeamentos são possíveis. A figura 6.12 apresenta o que seria o grafo de mapeamento correspondente a esse modelo, o que ilustra também a diferença de granularidade entre o modelo proposto e as soluções

usuais (que representam o software embarcado apenas como simples nodos nesse grafo). A figura 6.11(b) apresenta uma pequena variação da solução anterior (figura 6.11(a)), em que tanto o *softcore* sintetizado no PLD quanto a arquitetura física estão no mesmo nodo.

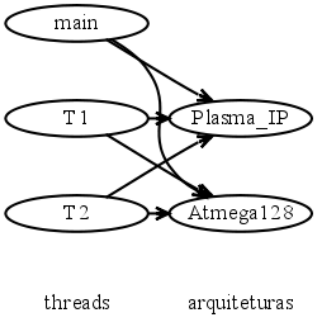


Figura 6.12: *Grafo de mapeamento correspondente ao M3EP apresentado. Muitas abordagens utilizam um modelo simplificado para software e hardware, representando apenas as threads da aplicação e as arquiteturas nas quais elas podem executar. Todos os demais componentes de software e de hardware são ignorados.*

Cromossomos de software são compostos por uma região promotora e por genes que representam componentes de software. Cada gene é composto por códons que incluem as configurações do componente (`traits`), suas dependências (`dependencies`), características funcionais (`functionalFeatures`), incluindo as próprias configurações e dependências de cada característica funcional (ver figura 5.4 na página 112), e que correspondem aos elementos que podem ser explorados em cada componente[4]. Mesmo que um componente não possua qualquer elemento que possa ser explorado, ele é incluído no cromossomo, pois faz parte da solução final. Componentes de software também podem ser mapeados a diferentes arquiteturas, e a escolha final desse mapeamento também é parte da exploração do espaço de projeto. Porém, decidiu-se que o mapeamento de componentes de software não é codificado no gene de cada componente, mas na região promotora de cada cromossomo de software, o que significa que todos os

[4]Configurações, dependências e características funcionais existem em qualquer componente, cyber/lógico ou físico.

componentes de software desse cromossomo são mapeados à mesma arquitetura. Isso diminui a quantidade de soluções inviáveis, reduz a complexidade geral das soluções e auxilia na geração de cada imagem (binária) de software. Por executarem na mesma arquitetura, assume-se que os componentes de software do mesmo cromossomo se comunicam através de endereços de memória e, portanto, essa comunicação não é representada em nenhum cromossomo de interconexão de software.

Cromossomos de hardware sintetizável existem apenas nos nodos de soluções que possuem software mapeados em *softcores* ou então componentes físicos eletro-eletrônicos do tipo PLD. Esses cromossomos possuem uma região promotora e genes que representam ou componentes de hardware sintetizável, ou canais de interconexão (`channels`) entre componentes de hardware sintetizável. Cada gene que representa um componente de hardware sintetizável inclui os elementos do componente que podem ser explorados, que são os mesmos dos componentes de software (configurações, dependências e características funcionais, com suas próprias configurações e dependências). De forma análoga aos cromossomos de software, os componentes de hardware também podem ser mapeados a diferentes componentes físicos PLD, e esses possíveis mapeamentos são codificados na região promotora do cromossomo de hardware sintetizável, e não nos próprios genes, de forma que todos os componentes do mesmo cromossomo são sintetizados no mesmo PLD. Todavia, ao contrário dos componentes de software, a comunicação entre os componentes de hardware sintetizável não é trivial, e precisa ser adequadamente codificada.

A comunicação de componentes de hardware sintetizável é representada num cromossomo específico, que não contém componentes, mas genes que codificam canais de interconexão, que correspondem à interface de comunicação desses componentes (`HardwarePort`) através de pinos (`HardwarePortPin`) e barramentos (`HardwarePorBus`), como é apresentado no apêndice 5 (figura 5.7 na página 116). Na criação aleatória desses cromossomos que irão formar parte da população inicial, criam-se conexões entre *channels* dos componentes que devem satisfazer algumas regras básicas: (1) Conectar inicialmente barramentos, depois pinos independentes; (2) Não podem ser conectados dois ou mais barramentos/pinos de saída; (3) Não

podem ser conectados dois ou mais barramentos/pinos de tipos distintos[5]; (4) Todos os pinos mandatórios devem ser conectados[6]; e (5) Não podem ser conectados barramentos/pinos que tenham suas restrições (`constraints`) violadas se forem conectados.

Essas regras básicas eliminam os erros mais óbvios e evitam muitas soluções inviáveis[7], mas são triviais e insuficientes para gerar corretamente possivelmente centenas ou milhares de conexões entre componentes, formando o circuito de um suporte de hardware completamente funcional. De fato, não é esse o objetivo da criação aleatória desses cromossomos de conexão. Embora deseja-se que a criação aproxime-se de circuitos funcionais, é função do processo de otimização penalizar soluções inviáveis, promover sua variação e seleção das soluções mais promissoras, de modo a aproximar-se, gradativamente, do circuito de um suporte de hardware completamente funcional. Contudo, regras melhores precisam ainda ser desenvolvidas para melhorar as populações iniciais, e assume-se que os estimadores fornecerão base para identificação das inconsistências e melhor evolução dos circuitos pelo tratamento e correção dessas soluções utilizando as adaptações desenvolvidas, o que não faz parte do escopo desta tese.

A abordagem adotada para diminuir o problema da geração de suportes de hardware inválidos inclui o conceito de "*template de circuito*", que é uma agregação de componentes de hardware físico ou sintetizável, fixos e interconectados de maneira específica, e que podem ser reusados em diferentes circuitos e suportes de hardware. Os *templates de circuito*, como definidos aqui, fornecem uma solução elegante, mesmo que parcial, para a dificuldade de geração de circuitos válidos, e com vantagens adicionais. As principais são sua granularidade e "rigidez" ajustáveis. Num extremo, pode-se não utilizar *templates de circuito* e permitir a exploração completa de cada componente de hardware e de cada conexão. Embora bastante interessante e flexível, essa opção extrema costuma gerar muitas soluções inviáveis e não convergir a uma solução ótima, pois o espaço de exploração torna-se plano (ver figura 2.6(a) na página 40). No outro extremo, pode-se criar um único *template* para todo

[5]Todos os barramentos/pinos possuem um tipo, que pode ser: *VCC, Ground, GPIO, Analogic, Reset, Interrupt, Connection,* ou *Other*.

[6]Barramentos/pinos mandatórios são aqueles que devem estar conectados para que o componente opere adequadamente, e incluem, principalmente os dos tipos *VCC, Ground,* e *Reset*.

[7]Uma solução é considerada inviável quando viola regras de projeto ou possui inconsistências.

o suporte de hardware, fixando todos os componentes e suas conexões, e permitindo explorar apenas suas configurações (*traits*). Essa opção é mais restritiva em termos de exploração do espaço de projeto, mas garante a viabilidade e funcionamento da plataforma. De fato, ela corresponde ao que é sugerido pela metodologia PbD. Contudo, os *templates de circuitos* permitem abordagens intermediárias.

Templates podem representar agrupamentos recorrentes em circuitos, como configurações para fontes de alimentação, etapas de potência, interconexão processador/memória, conexões de dispositivos a barramentos, entre outros. Nesse sentido, o suporte de hardware pode ser visto como a interconexão de sub-circuitos, ou *templates de circuitos*. A estrutura e conexões internas de cada *template de circuitos* é fixa, mas a estrutura e conexão entre diferentes *templates de circuitos* pode ser explorada e evoluída. Além disso, também a "rigidez" dos *templates de circuitos* pode ser controlada usando outras estruturas desenvolvidas nesta tese. *Templates* podem ser cadastrados e mantidos completamente fixos ao desabilitar os operadores de variação sobre eles, utilizando suas regiões promotoras. Contudo, como é possível ter regiões promotoras em todos os elementos genéticos, é possível permitir também que alguns elementos do *template* possam ser mais ou menos que fixos que outros. Com isso, pode-se, por exemplo, fixar toda a estrutura de um *template*, mas permitir que um componente específico possa ser trocado por outro. Por fim, como todos os cromossomos das soluções não-dominadas são mantidos como saídas do projeto de sistema embarcado (ver figura 5.1 na página 103), suportes de hardware podem ser reusados em futuros projetos, ou partes desses cromossomos podem ser fixadas, criando um novo *template de circuito*.

Componentes físicos eletro-eletrônicos, ou simplesmente componentes de hardware físico, existem em toda solução de sistema embarcado, e eles são codificados nos cromossomos de hardware físico. De modo análogo aos demais cromossomos, ele possui uma região promotora e genes que representam os elementos exploráveis dos componentes de hardware físico ou então seus canais de interconexão. Diferente dos outros níveis, o hardware físico não precisa ser mapeado em outro tipo de componente[8]. A interconexão dos

[8]Nesta tese, componentes físicos mecânicos tiveram uma modelagem mínima, apenas para permitir que ela possa ser refinada futuramente. Nesse sentido, pode-se imaginar como um primeiro aperfeiçoamento básico, o mapeamento de componentes físicos a uma plataforma mecânica que os suporte, como uma placa de circuito impresso.

componentes de hardware físico considera ainda outras regras, como a compatibilidade das características elétricas (`ElectricalFeature`) de cada barramento ou pino. Como as características elétricas incluem, basicamente, tensões e correntes mínimas, desejadas e máxima, além da impedância, e que essas informações podem depender de configurações do componente, as restrições (`constraints`) são a melhor forma de definir regras flexíveis e expansíveis para interconexão de componentes. Uma representação completa das características eletro-eletrônicas dos componente foge do escopo desta tese.

6.4 Modelo da Estrutura de um Indivíduo

A visão geral do modelo computacional do indivíduo desse novo modelo evolucionário com adaptações para a otimização multiobjetivo de problemas complexos tem sua estrutura geral representada pela figura 6.13. Um indivíduo é representado pela classe `Individual`, e é composto por um genótipo (interface `GenotypeInterface`), que representa sua estrutura interna, por um fenótipo (interface `PhenotypeInterface`), que representa essa solução no espaço de soluções, e por aptidões (classe `ObjectivesMap`), que mapeiam o fenótipo a pontos no espaço de decisões. Esse modelo altera a estrutura da representação de soluções dos problemas sendo otimizados (indivíduos) e, consequentemente, dos operadores genéticos sobre essa estrutura.

Fenótipo

O fenótipo do indivíduo corresponde à tradução do genótipo no contexto do problema sendo tratado, ou seja, valores no espaço de soluções. Assim, o fenótipo é dependente do problema e pode assumir diferentes formas. Por isso, as muitas possíveis traduções não serão apresentadas. Em problemas de otimização multiobjetivo, a tradução mais comum é a de um vetor de M números reais, que correspondem à avaliação das M funções-objetivo $f_i(\overrightarrow{x})$ após o mapeamento do genótipo do indivíduo nas variáveis $\overrightarrow{x}$ do problema $(\mathfrak{m}(g_i) \mapsto x_i)$. Em problemas mais complexos, o fenótipo deve representar uma solução ao problema. No caso do projeto de sistemas embarcados, o fenótipo deve representar um sistema embarcado projetado. Quando usando a metodologia ADESD, isso significa a especificação de um conjunto

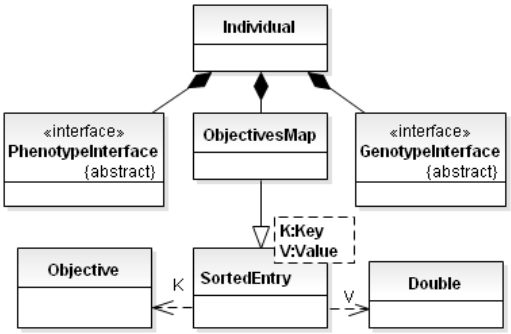


Figura 6.13: *Modelo de um indivíduo. O indivíduo é composto por um genótipo, um fenótipo e um conjunto de mapeamentos de objetivos, que associam um número real a cada objetivo de otimização, que corresponde a um ponto no espaço de decisão.*

.

de componentes de software, de hardware sintetizável e de hardware físico, suas configurações, características funcionais, mapeamentos e especificação das ferramentas de compilação e síntese dos componentes lógicos.

Objetivos

O fenótipo de um indivíduo deve ser avaliado em relação aos objetivos de otimização e um valor real deve ser mapeado a cada objetivo (classe `Objective`), que representa uma função-objetivo f_i. A classe `ObjectivesMap` faz esse mapeamento, ou seja, associa um valor real como resultado da avaliação de uma solução a cada objetivo a ser otimizado. Cada objetivo (`Objective`) possui, basicamente, uma descrição informal, se deve ser maximizado ou minimizado e valores máximo e mínimo que o valor real associado a ele pode assumir ($f_{i,max}, f_{i,min}$), que podem ser usados para a normalização desse objetivo (ver equação 2.6 na página 26). Por agregar os resultados de todos os objetivos de uma solução, ele agrega uma classe que especifica uma forma de dominância, que é responsável por determinar se os valores de um conjunto de objetivos dominam ou são dominados por outro conjunto de valores, conforme alguma forma de dominância escolhida para representar as soluções preferíveis (seção 2.2).

Genótipo

A figura 6.14 apresenta a visão geral do modelo do genótipo. A interface `GenotypeInterface` pode ser realizada (implementada) por diferentes classes em diferentes composições, permitindo muitas diferentes possíveis estruturas (epi)genéticas do indivíduo. De modo geral, um genótipo é composto por cromossomos (uma realização da interface `CromossomeInterface`), que são compostos por locus (uma realização da interface `LocusInterface`), que são compostos por genes (uma realização da interface `GeneInterface`), que são compostos por códons (uma realização da interface `CodonInterface`), que representam as unidades fundamentais de informação. Cada uma dessas estruturas pode ainda incluir uma região promotora (uma realização da interface `PromotingRegionInterface`, mostrada apenas na figura 6.16), que permite alterar seu comportamento por diversos fatores, possivelmente epigenéticos. Uma das contribuições neste modelo é que o gene pode não ser a estrutura final de informação genética, mas sim um conjunto de informações correlacionadas, o que auxilia no tratamento da epistasia e na melhor representação de elementos mais complexos, como os componentes em sistemas embarcados. A separação criteriosa de conceitos traz outros benefícios e contribuições, como a possibilidade de especificar genes heterogêneos, que agrupam informações de diferentes tipos, como variáveis reais, discretas, links, e listas, entre outras. Essa característica é particularmente útil para representação de componentes de sofware e hardware, que são parametrizados e configurados por variáveis de diferentes tipos.

A figura 6.15 apresenta o modelo do códon que é, em última instância, quem representa as variáveis codificadas do problema. Códons guardam variáveis codificadas, que podem ser variáveis inteiras ou reais nos problemas que são modelados matematicamente por funções-objetivo a serem minimizadas ou maximizadas. Todavia, quando o problema pode ser modelado como um conjunto de equações matemáticas, são necessárias outras formas de representar suas variáveis. Pode-se facilmente impor limites (*bounds*) a essas variáveis, o que garante que seus valores estarão sempre entre os limites superior e inferior que são fornecidos. Entretanto, essa estrutura, já bastante comum, não é suficiente ou adequada para representar outros tipos de possíveis variáveis em problemas sendo otimizados. Inspirados na GEP (GEP - *Gene Expression Programming*), nesta tese adaptei a estrutura de um gene GEP para representar outros tipos de variáveis.

Uma necessidade na representação de sistemas embarcados dirigidos

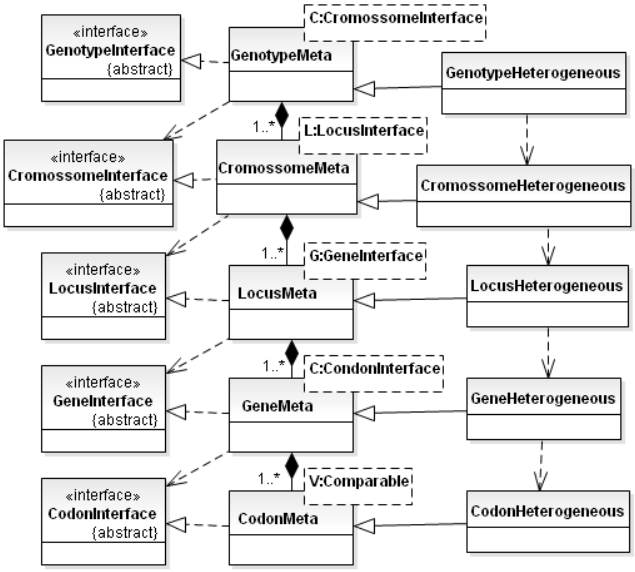


Figura 6.14: *Modelo do genótipo de um indivíduo. Apresenta algumas das diferentes possíveis interfaces e realizações dessas interfaces que podem ser utilizadas para compor a estrutura genotípica de um indivíduo.*

pela aplicação é o de variáveis que modelem (e possam evoluir) funções-objetivo e outras equações, variáveis discretas, dependências entre componentes, mapeamentos entre componentes de software e arquiteturas, mapeamentos entre componentes de hardware sintetizável e PLDs, e conexões entre componentes de hardware para formar um suporte de hardware. Nesta tese, variações dos genes da GEP foram desenvolvidas como códons que modelam esses elementos. Uma visão geral do gene GEP (aqui modelado como um códon) usado para representar equações foi apresentado na tabela 3.1. Embora simplificada, aquela visão do gene GEP destaca a impossibilidade de geração de soluções inválidas, característica que é preservada nas adaptações desenvolvidas para suportar as necessidades de projetos de sistemas embarcados. A classe `CodonDiscrete` é a base para qualquer códon que codifique variáveis discretas, onde apenas um valor de uma lista de possíveis valores pode ser codificado. Essa classe utiliza a estrutura de constantes para representar

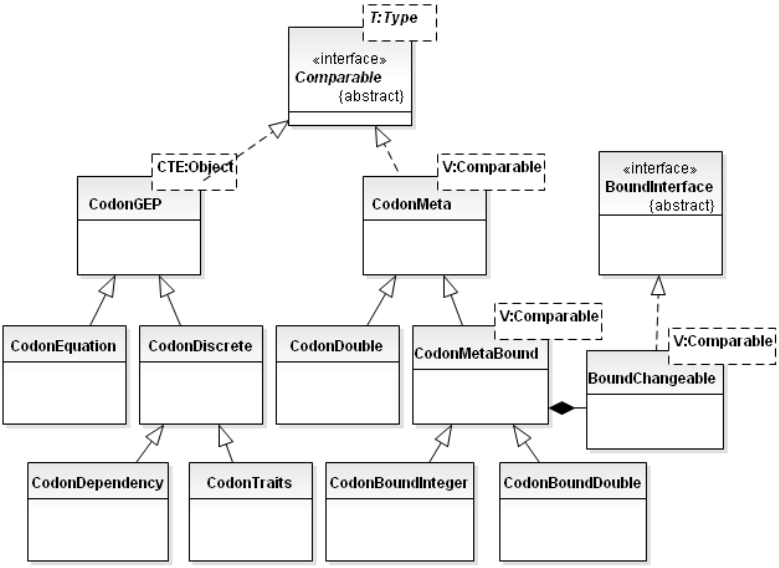


Figura 6.15: *Modelo do códon de um gene. Diferentes tipos de variáveis podem ser codificadas e manipuladas. Além das variáveis reais usuais, variáveis discretas e "links" para outros elementos genéticos foram representadas, permitindo a associação entre genes e cromossomos, por exemplo.*

os possíveis valores da variável discreta e fixa o cabeçalho (*header h*) ao elemento "?", que especifica uma constante, transformando a especificação de uma equação GEP em uma variável discreta. Mecanismos similares foram desenvolvidos para transformar a estrutura de constantes em "ponteiros" para genes, o que permite representar, entre outras coisas, a dependência alternativa entre componentes. A inclusão de mais algumas restrições, em conjunto com a inclusão de elementos epigenéticos (descritos a seguir) permite fixar "ponteiros" entre tipos específicos de genes, o que permite representar o mapeamento entre componentes de software e arquiteturas, e entre componentes de hardware sintetizável e dispositivos lógicos programáveis.

A visão geral do modelo computacional da região promotora é apresentada na figura 6.16. Basicamente, ela é uma tabela de espalhamento que permite buscar *valores* com base em *chaves de acesso*. As *chaves de acesso*

são, em geral, *Strings* ou objetos definidos e conhecidos pelas classes que podem alterar o comportamento dos elementos genéticos, que são (1) Os operadores (epi)genéticos; (2) O tradutor de genótipo em fenótipo; (3) O avaliador do fenótipo; e (4) O controle do algoritmo evolucionário em si. Os *valores* acessados através das *chaves* são objetos diversos, dependendo da classe que está acessando a região promotora. Dentre as várias opções possíveis, a figura 6.16 apresenta a interface `OperatorParameterInterface`, que representa o objeto comumente recuperado pelos operadores quando seu comportamento é distinto para aquele elemento. Outra opção apresentada nessa figura é a interface `IndicatorInterface`, que representa indicadores de qualidade do algoritmo de otimização, e que pode ser usado, por exemplo, para alterar o gene com base em indicadores como a diversidade da população ou a pressão de seleção.

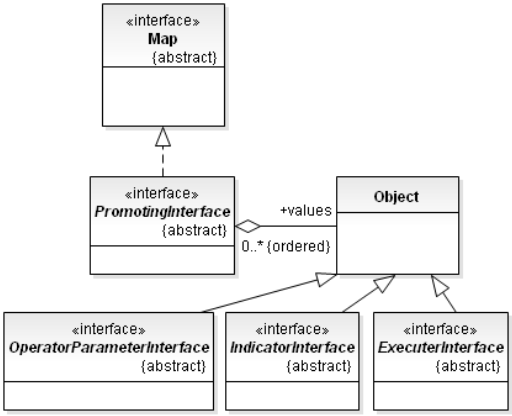


Figura 6.16: *Modelo da região promotora. A região promotora pode preceder qualquer estrutura genética, alterando seu comportamento. Ela corresponde a uma estrutura de busca baseada em chaves de acesso, que são conhecidas pelos elementos que acessam o genótipo. Os valores buscados descrevem como a estrutura correspondente deve ser interpretada ou alterada.*

Para especificação de problemas passíveis de otimização (MOP), uma solução é codificada num indivíduo usando o modelo previamente apresentado, formando a população inicial. O mecanismo evolucionários dos EA e a otimização multiobjetivo são responsáveis por gerar variações nessas solu-

ções, guiando-as para soluções ótimas. Assim, o próximo capítulo apresenta a avaliação da qualidade do modelo evolucionário apresentado neste capítulo.

CAPÍTULO 7

AVALIAÇÃO DE QUALIDADE DO MODELO EVOLUCIONÁRIO

É indesejável acreditar numa proposição quando não há a menor base para supô-la verdadeira.
—BERTRAND RUSSEL (Sceptical Essays, 1928)

É comum supor que a massa de nossas crenças provém de alguma base racional, e que o desejo é apenas uma força perturbadora ocasional. Exatamente o oposto se aproximaria mais da verdade: a grande massa de crenças pela qual somos amparados em nossa vida diária é apenas projeção do desejo, corrigida aqui e ali, em pontos isolados, pelo rude choque dos fatos.
—BERTRAND RUSSEL (Sceptical Essays, 1928)

Este capítulo apresenta a avaliação da qualidade de otimização do modelo evolucionário de otimização. O capítulo começa explicitando quais informações se deseja extrair dos experimentos e como os experimentos foram planejados, executados e avaliados. As seções seguintes descrevem, cada uma, um experimento específico, incluindo seus objetivos, método de execução e resultados. A última seção deste capítulo (7.6) sintetiza os resultados e discussões.

7.1 Descrição dos Experimentos

A avaliação do modelo evolucionário com adaptações de inspiração biológica, apresentado no capítulo 6, é feita em termos de sua eficiência na otimização multiobjetivo. Com isso, é possível verificar o quão próximas o ótimo estariam sistemas embarcados gerados a partir da exploração do espaço de projeto usando as mesmas adaptações em seu modelo evolucionário. Essa eficiência na otimização é quantificada ao submeter cada adaptação do modelo a um conjunto de problemas de teste conceituado e obtendo, para cada problema, o valor numérico de um indicador de qualidade completo e

compatível. Dado que cada adaptação é ortogonal às demais, cada uma delas possa ser avaliada de forma independente. O procedimento geral para avaliação de uma adaptação do modelo evolucionário pode ser descrito com as seguintes etapas gerais:

1. A adaptação a ser avaliada é incluída no modelo de otimização;
2. Para cada problema de teste são executadas uma pré-otimização para determinação dos limites mínimo e máximo de cada objetivo, e então 30 (trinta) otimizações (chamadas aqui de *execuções*), com populações iniciais aleatórias diferentes;
3. Para cada *execução* são anotados os valores do indicador de qualidade e das soluções não-dominadas;
4. Após as *execuções* de um problema de teste, os valores de média aritmética, desvio-padrão, mínimo e máximo são anotados e correspondem aos resultados de um uma configuração específica;
5. Os valores são apresentados em gráficos e tabelas.

Para a avaliação da qualidade de cada adaptação realizada, para cada problema de teste e para cada otimização, o indicador de hipervolume ponderado é calculado. O cálculo desse indicador exige a especificação de alguns parâmetros, destacando-se o ponto de referência, o ponto ideal e o ponto de nadir. Como eles são dependentes dos valores das funções-objetivos de cada problema, é necessário realizar uma pré-otimização de ajuste. Nessa pré-otimização, os valores de mínimo e máximo de cada função-objetivo são anotados e incluídos como parâmetros do otimizador para realizar a transformação de escala das funções-objetivos (ver equação 2.6) para o intervalo $\Re[0,1]$, visando eliminar a influência de diferentes escalas no hipervolume. Com isso, define-se os pontos ideal e de referência nas coordenadas $\Re^m = (0,0,...,0)$, pois o limite mínimo de cada função-objetivo normalizada é 0 (zero), e o ponto de nadir nas coordenadas $\Re^m = (2.1,2.1,...,2.1)$, apesar do limite superior normalizado ser 1.0, conforme sugerido em [Bader 2009]. Outro parâmetro do hipervolume ponderado que merece destaque é a distribuição de probabilidade que funciona como "ponderação" sobre as funções-objetivos e que é utilizada para representar as preferências do projetista [Bader 2009]. Embora essa possibilidade seja muito

útil aos projetos reais, nas avaliações realizadas e descritas neste capítulo, não foi utilizada nenhuma distribuição de pesos para não inserir um *bias* nos resultados.

A não ser quando for explicitamente informado num experimento específico o caso contrário, todos os experimentos foram realizados no mesmo microcomputador[1] usando:

- Algoritmos evolucionários;
- Operador de variação de recombinação SBX de 1-ponto;
- Operador de mutação com distribuição polinomial;
- Codificação com números reais;
- Estratégia de seleção NSGA2 (torneio 0);
- Dominância de Pareto; e
- Critério de parada baseado unicamente na quantidade máxima de gerações.

Os algoritmos evolucionários, a não ser quando explicitamente informado o caso contrário, utilizam os parâmetros extraídos de [Bader 2009] (N, α, *execuções*) e de [Deb et al. 2001] ($\varphi, \zeta, \eta_{rec}, \eta_{mut}$), permitindo uma comparação mais direta, sendo eles:

- Número máximo de gerações (N): 200;
- População (α): 50;
- Indivíduos-pais por geração (μ): 20;
- Indivíduos-filhos por geração (λ): 20;
- Recombinação ($\eta_{rec} = 15$) com taxa de aplicação (φ):1.0;
- Mutação ($\eta_{mut} = 20$) com taxa de aplicação (ζ): 1/n, onde n é o número de variáveis do problema.

As próximas seções descrevem em detalhes os problemas de teste utilizados e os experimentos de avaliação que foram realizados. Todos eles seguem o procedimento básico descrito nesta seção, e possíveis diferenças são explicitadas na descrição do experimento em que ocorrem. De maneira geral, os experimentos realizados buscam avaliar a qualidade de otimização dos seguintes elementos e adaptações:

1. Otimizador básico (seção 7.3): avalia se o otimizador, sem qualquer

[1] Microcomputador com processador Intel®Core™2 Duo T5500 (1.66 GHz, 667 MHz FSB, 2 MB L2 cache), Mobile Intel®GraphicsMedia Accelerator 950, com 2.47GB DDR2 RAM, e HDD 160GB 7200 rpm, com sistema operacional Linux Ubuntu, kernel 2.6.24-24.

alteração, apresenta desempenho compatível aos de outras pesquisas da literatura. Serve apenas para validar a base de comparação dos demais experimentos.

2. Auto-adaptação de operadores genéticos (seção 7.4): avalia a auto-adaptação de operadores genéticos de variação de forma diferenciada para cada códon específico, usando as regiões promotoras.

3. Neutralidade genética e multiploidismo (seção 7.5): avalia se o multiploidismo e a consequente neutralidade genética auxiliam na convergência da otimização.

Para cada experimento, são apresentadas as seguintes informações: (1) As fronteiras de Pareto do otimizador sendo avaliado com a fronteira de Pareto ótima, ou mais especificamente, a aproximação do conjunto de Pareto produzida pelo otimizador sob teste em relação ao conjunto completo de Pareto; (2) Os valores do indicador de hipervolume correspondente a cada fronteira, o que permite quantificar a área por ela dominada; (3) A evolução do indicador de hipervolume ao longo das iterações, o que permite avaliar a convergência da otimização; e (4) Uma comparação entre o indicador de hipervolume do otimizador sendo avaliado com um indicador de referência.

7.2 Problemas de Teste

As adaptações realizadas no modelo evolucionário para exploração do espaço de projetos foram submetidas a problemas de teste para avaliação de sua qualidade. Para ser possível verificar a proximidade e distribuição das soluções encontradas em relação à fronteira de Pareto ótima, é necessário que a equação que descreve a fronteira de Pareto dos problemas de testes seja conhecida, o que não ocorre com o projeto de sistemas embarcados reais. Além disso, também é necessário que os problemas de teste apresentem diferentes obstáculos à otimização, o que permite avaliar a eficiência dos diferentes algoritmos e/ou adaptações realizadas. Por isso, não foram utilizados problemas reais de exploração do espaço de projeto em sistemas embarcados específicos, pois nesse caso não seriam conhecidas as equações das funções-objetivos, nem as equações das fronteiras de Pareto e nem quais obstáculos eles oferecem aos algoritmos. Portanto, também não seria possível saber o

quão próximas do ótimo estariam as soluções. Conjuntos de testes "artificiais" possuem vantagens sobre problemas reais no que se refere à aplicação dos algoritmos testados [Huband et al. 2006]. Assim, foram usados problemas de teste consagrados na literatura, com fronteira de Pareto conhecida e que oferecem obstáculos específicos.

O conjunto de problemas de testes utilizado para avaliação do modelo evolucionário é o ZDT [TIK 2008], sigla de seus autores: Zitzler, Deb e Thiele, e é composto por 6 problemas de teste, descritos a seguir.

ZDT1

O problema ZDT1 é definido pela equação (7.1), cuja fronteira de Pareto é definida pela equação (7.2) e apresentada na figura 7.1. Esse problema tem 30 variáveis no intervalo $\Re[0,1]$ e fronteira de Pareto convexa, contínua e com uma distribuição uniforme das soluções sobre a fronteira. É o problema mais fácil de ser otimizado e o único obstáculo inserido é o tratamento de um grande número de variáveis de decisão (30 variáveis).

$$\begin{aligned} &Minimizar: f_1(x) = x_1, \\ &Minimizar: f_2(x) = g\left(1.0 - \sqrt{\frac{f_1}{g}}\right), \\ &\qquad g(x_2,...,x_n) = 1.0 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\ &Sujeito\ a: 0 \leq x_i \leq 1, \forall i = 1,2,...,n \end{aligned} \tag{7.1}$$

$$x_2 = 1 - \sqrt{x_1} \tag{7.2}$$

ZDT2

O problema ZDT2 é definido pela equação (7.3), cuja fronteira de Pareto é definida pela equação (7.4) e apresentada na figura 7.2. Esse problema tem 30 variáveis no intervalo $\Re[0,1]$ e uma fronteira de Pareto não-

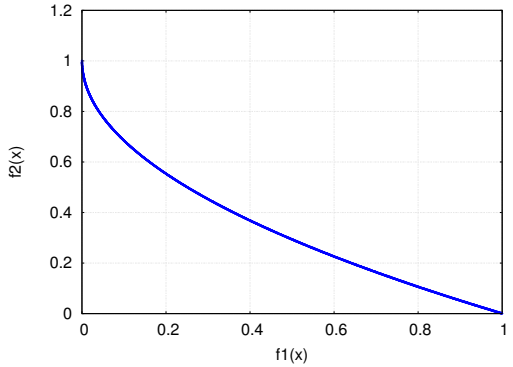


Figura 7.1: *Fronteira de Pareto do problema de teste ZDT1. Esse problema tem fronteira convexa, contínua e uma distribuição uniforme das soluções sobre a fronteira. É o problema mais fácil de ser otimizado.*

convexa, o que causa dificuldade a alguns algoritmos.

$$\begin{aligned} &Minimizar: f_1(x) = x_1, \\ &Minimizar: f_2(x) = g\left[1 - \left(\frac{x_1}{g(x)}\right)^2\right], \\ &g(x) = 1.0 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\ &Sujeito\ a: 0 \leq x_i \leq 1, \forall i = 1,2,...,n \end{aligned} \tag{7.3}$$

$$x_2 = 1 - x_1^2 \tag{7.4}$$

ZDT3

O problema ZDT3 é definido pela equação (7.5), cuja fronteira de Pareto é desconexa e definida pela equação (7.6) e apresentada na figura 7.3. Esse problema tem 30 variáveis no intervalo $\Re[0,1]$, fronteiras de Pareto desconexas com uma distribuição uniforme das soluções nas fronteiras. A dificuldade imposta aos algoritmos consiste em manter populações distintas em

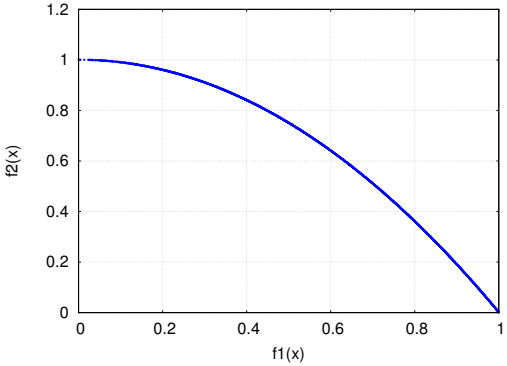


Figura 7.2: *Fronteira de Pareto do problema de teste ZDT2. Esse problema tem uma fronteira não-convexa, o que causa dificuldade a alguns algoritmos.*

cada trecho desconexo da fronteira.

$$\begin{aligned} &Minimizar: f_1(x) = x_1, \\ &Minimizar: f_2(x) = g(x)\left[1 - \sqrt{\frac{x_1}{g(x)}} - \frac{x_1}{g(x)} sin(10\pi x_1)\right] \\ &\qquad g(x) = 1.0 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\ &Sujeito\ a: 0 \leq x_i \leq 1, \forall i = 1,2,...,n \end{aligned} \tag{7.5}$$

$$\begin{aligned} &(x_1, x_2), x_i \in F, x_2 = 1 - \sqrt{x_1} - x_1.sin(10\pi x_1) \\ &\quad F = [0.0, 0.0830015349] \cup \\ &\qquad (0.1822287280, 0.2577623634] \cup \\ &\qquad (0.4093136748, 0.45538821041] \cup \\ &\qquad (0.6183967944, 0.6525117038] \cup \\ &\qquad (0.8233317983, 0.8518328654] \end{aligned} \tag{7.6}$$

ZDT4

O problema ZDT4 é definido pela equação (7.7), cuja fronteira de Pareto é definida pela equação (7.8) e apresentada na figura 7.4. Esse problema tem 10 variáveis, com x_1 no intervalo $\Re[0,1]$ e as demais no intervalo

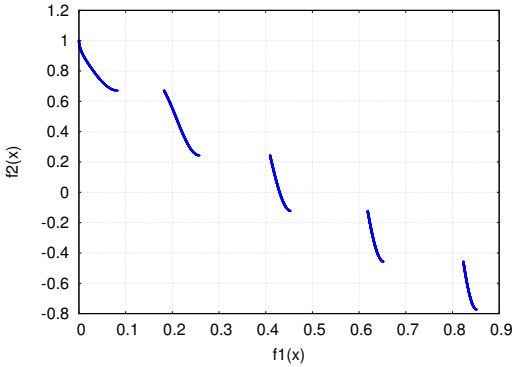


Figura 7.3: *Fronteira de Pareto do problema de teste ZDT3. Esse problema tem fronteiras desconexas com uma distribuição uniforme das soluções nas fronteiras. A dificuldade está em manter populações distintas em cada trecho da fronteira.*

[-5,5]. Possui fronteira convexa, mas com 100 fronteiras locais, o que exige dos algoritmos ultrapassar muitos mínimos locais até encontrar a fronteira de Pareto ótima. Embora a fronteira pareça idêntica ao problema ZDT1, os problemas são diferentes e ZDT4 apresenta maior dificuldade aos otimizadores, devido às fronteiras locais que não aparecem na figura.

$$\begin{aligned} &Minimizar : f_1(x) = x_1, \\ &Minimizar : f_2(x) = g\left[1 - \left(\frac{x_1}{g(x)}\right)^2\right], \\ &\qquad g(x) = 1.0 + 10(n-1)\sum_{i=2}^{n}(x_i^2 - 10cos(4\pi x_i)) \\ &Sujeito\ a : 0 \leq x_1 \leq 1 \\ &\qquad -5 \leq x_i \leq -5, \forall i = 2, ..., n \end{aligned} \tag{7.7}$$

$$x_2 = 1 - \sqrt{x_1} \tag{7.8}$$

ZDT5

O problema ZDT5 é definido pela equação (7.9), cuja fronteira de

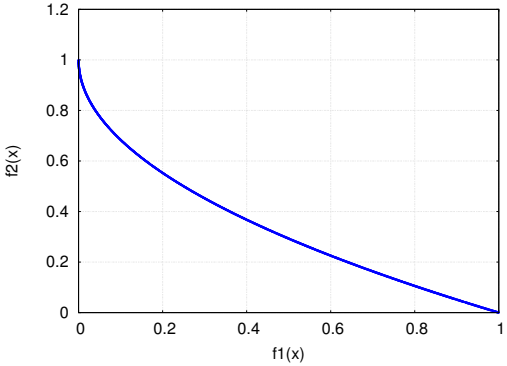


Figura 7.4: *Fronteira de Pareto do problema de teste ZDT4. Este problema tem fronteira convexa, mas com 100 fronteiras locais, causando uma dificuldade aos algoritmos encontrarem a fronteira ótima.*

Pareto é definida pela equação (7.10) e apresentada na figura 7.5. Esse problema é definido sobre strings de bits, em que x_1 é representado com 30 bits e as demais variáveis com 5 bits cada. Esse problema possui 1023 fronteiras locais, o que casa dificuldade aos algoritmos e também porque esse é um problema deceptivo e sua codificação é distinta de variáveis reais.

$$\begin{aligned} &Minimizar: f_1(x) = x_1, \\ &Minimizar: f_2(x) = g\left(1.0 - \sqrt{\frac{f_1}{g}}\right), \\ &\qquad g(x_2,...,x_n) = 1.0 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\ &Sujeito\ a: 0 \leq x_i \leq 1, \forall i = 1,2,...,n \end{aligned} \tag{7.9}$$

$$x_2 = 1 - \sqrt{x_1} \tag{7.10}$$

ZDT6

O problema ZDT6 é definido pela equação (7.11), cuja fronteira de Pareto é definida pela equação (7.12) e apresentada na figura 7.6. Esse

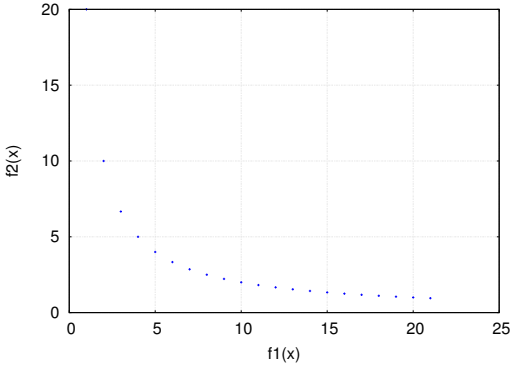


Figura 7.5: *Fronteira de Pareto do problema de teste ZDT5. Esse problema é definido sobre strings de bits e há 1023 fronteiras locais, o que casa dificuldade aos algoritmos e também porque esse é um problema deceptivo.*

problema tem 10 variáveis no intervalo $\Re[0,1]$, uma fronteira de Pareto não-convexa com soluções concentradas numa pequena área da fronteira de Pareto, causando duas dificuldades aos algoritmos.

$$\begin{aligned} &Minimizar: f_1(x) = 1 - e^{-4x_1} sin^6(6\pi x_1), \\ &Minimizar: f_2(x) = 1 - \left(\frac{f_1(x)}{g(x)}\right)^2, \\ &g(x) = 1 + 9\left[\frac{\sum_{i=2}^{n} x_i}{n-1}\right]^{0.25} \\ &Sujeito\ a: 0 \leq x_i \leq 1, \forall i = 1,2,...,n \end{aligned} \tag{7.11}$$

$$x_2 = 1 - x_1^2,\ x_1 \in [0.2807753191, 1.0] \tag{7.12}$$

7.3 Avaliação Geral do Otimizador

Este experimento avalia a eficiência do otimizador básico medida através do indicador de hipervolume ponderado sobre o conjunto de testes

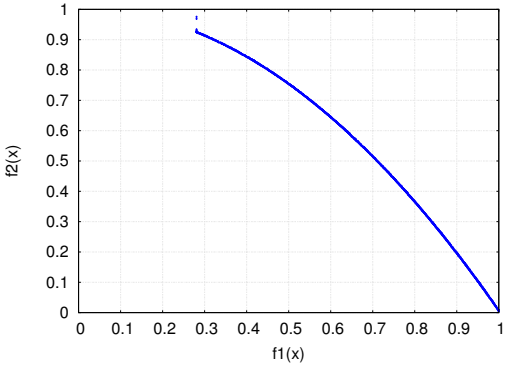


Figura 7.6: *Fronteira de Pareto do problema de teste ZDT6. Esse problema tem uma fronteira não-convexa com soluções concentradas numa pequena área da fronteira de Pareto, causando duas dificuldades aos algoritmos.*

ZDT, sem a inclusão de qualquer adaptação. Ressalta-se que só é possível uma comparação direta com outras pesquisas que utilizam o mesmo indicador e os mesmos parâmetros do algoritmo evolucionário, embora a apresentação de gráficos com a fronteira de Pareto e as soluções encontradas também permitam uma comparação geral de sua eficiência e comparação com outros trabalhos. Além disso, como as fronteiras de Pareto desse conjunto de testes são conhecidas, também é possível comparar a eficiência de cada adaptação em relação ao ótimo.

Assim, a avaliação inicia com o cálculo do indicador de hipervolume sobre a função da fronteira de Pareto para cada problema de teste do *benchmark* ZDT, de forma a obter o maior hipervolume dominado possível para cada problema, ou seja, o valor ótimo. Um otimizador que gera esse conjunto de soluções, ou um conjunto de soluções que domine o mesmo hipervolume, é "ótimo" e passa a ser denominado aqui como $MOEA_{OTM}$. Ressalta-se que o hipervolume dominado depende dos pontos ideal, de referência e de nadir que são especificados para o indicador de hipervolume. Em todos os experimentos feitos, esses pontos são aqueles especificados na seção anterior, ou seja, ponto de referência (0.0, 0.0), ponto ideal (0.0, 0.0) e ponto de nadir (2.1, 2.1).

O indicador de hipervolume I^w_{HOTM} calculado para o $MOEA_{OTM}$ é

apresentado na tabela 7.1, e representa a área dominada pela fronteira de Pareto, que é o limite máximo obtido por qualquer otimizador. Esse não é um limite utópico e pode ser obtido por qualquer bom otimizador que execute com uma população grande o suficiente por uma quantidade de gerações grande o suficiente, além de parâmetros de configuração adequados. Como as fronteiras de Pareto dos diferentes problemas de teste possuem escalas diferentes, e como deseja-se avaliar o efeito de otimizadores que terão desempenho menor que o ótimo e, portanto, escalas um pouco maiores, os valores do indicador de hipervolume apresentados na tabela 7.1 foram calculados a partir dos valores normalizados das fronteiras de Pareto para o intervalo $\Re[0.0, 0.67]$, de forma que a escala dos otimizadores que serão avaliados (com as adaptações desenvolvidas), provavelmente ainda estejam no intervalo $\Re[0.0, 2.1]$, devido ao limite superior imposto pelo ponto de nadir.

Tabela 7.1: *Eficiência do otimizador ótimo com o benchmark ZDT. A eficiência é medida com o indicador de hipervolume ponderado, obtido a partir da função que define a fronteira de Pareto para cada problema.*

I^w_{HOTM}	Problema de Teste					
	ZDT1	ZDT2	ZDT3	ZDT4	ZDT5	ZDT6
Valor	4,1875	3,9655	4,0336	4,1876	4,3189	4,0348

Conhecendo o valor ótimo, executou-se um experimento com o otimizador básico, sem qualquer adaptação. Por isso, esse otimizador passa a ser referenciado como $MOEA_{REF}$. Este experimento seguiu o procedimento geral apresentado na seção 7.1, com os mesmos parâmetros especificados.

A figura 7.7 mostra fronteira de Pareto resultante do conjunto de soluções não-dominadas obtido pelo $MOEA_{REF}$, apresentada juntamente com a fronteira de Pareto ótima, para cada problema. Nessa figura, a fronteira ótima corresponde à função que determina a fronteira de Pareto. A proximidade da fronteira do conjunto de soluções não-dominadas identifica a qualidade da otimização, o que é mensurado quantitativamente pelo hipervolume. Pela análise da figura 7.7 percebe-se que o $MOEA_{REF}$ não consegue, com os parâmetros dados, gerar um conjunto de soluções não-dominadas muito próximo da fronteira ótima. Como já era esperado, o problema ZDT1 foi o mais fácil de otimizar (as duas fronteiras estão mais próximas). A fronteira não-convexa do problema ZDT2 causou dificuldade em parte do conjunto de soluções; As

soluções ao problema ZDT4 ficaram presas num ótimo local e a não uniformidade da distribuição da fronteira do ZDT6 fez com que as soluções encontradas ficassem distantes do ótimo. Esses resultados eram esperados, uma vez que os parâmetros estabelecidos não permitem uma convergência completa ao conjunto de soluções de Pareto.

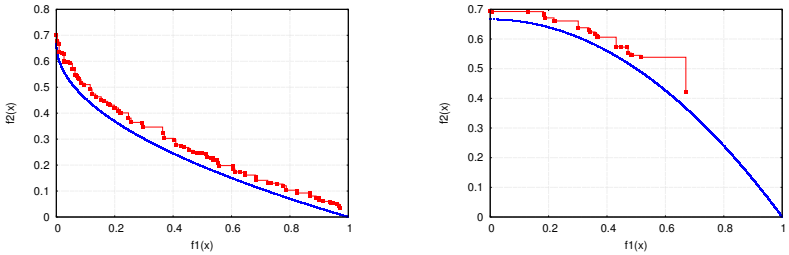
O valor final do indicador de hipervolume para cada um dos problemas de teste com o otimizador básico $MOEA_{REF}$ é apresentado na tabela 7.2. Essa mesma tabela apresenta também a relação entre o indicador do otimizador básico e o indicador ótimo (da tabela 7.1), ou seja, I^w_{HREF}/I^w_{HOTM}. Esse valor é apresentado em percentual e representa o quão melhor ou pior foi a eficiência da otimização em relação ao ótimo[2]. Conforme essa tabela, percebe-se que o $MOEA_{REF}$ teve seu melhor desempenho com os problemas ZDT3 e ZDT1, que foram apenas 1,785% e 1,96% inferiores ao ótimo, respectivamente, e teve seu pior desempenho no problema ZDT6, que foi 45,495% inferior ao ótimo (100% - 54,504%).

Tabela 7.2: *Eficiência do otimizador básico com o benchmark ZDT. A eficiência é medida com o indicador de hipervolume ponderado, obtido a partir do conjunto de soluções não-dominadas para cada problema. A relação com o ótimo também é apresentada.*

	Problema de Teste					
	ZDT1	ZDT2	ZDT3	ZDT4	ZDT5	ZDT6
I^w_{HREF}	4,1054	3,3899	3,9616	3,6339	4,0367	2,1991
$\% \frac{I^w_{HREF}}{I^w_{HOTM}}$	98,040	85,485	98,215	86,779	93,466	54,505

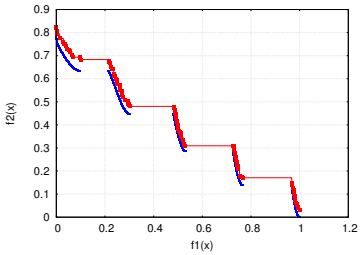
Como todos os algoritmos podem, enfim, convergir completamente ao conjunto de soluções de Pareto, ou seja, podem terminar com o mesmo valor do indicador de hipervolume, é importante analisar sua eficiência nesse processo. A convergência do algoritmo pode ser avaliada pelo aumento da área dominada pelas melhores soluções obtidas a cada iteração da otimização, ou seja, a evolução do indicador de hipervolume ao longo das gerações.

[2] 100% significa desempenho idêntico; Valores maiores que 100% indicam desempenho superior e valores menores que 100% indicam desempenho pior. Por exemplo, o valor 97% significa que a adaptação testada teve desempenho de 97% do desempenho da referência, ou seja, foi 3% pior.

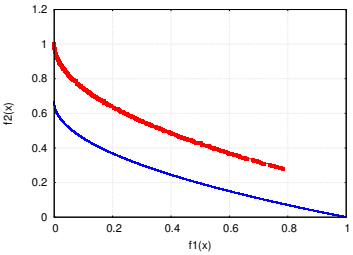


(a) ZDT1

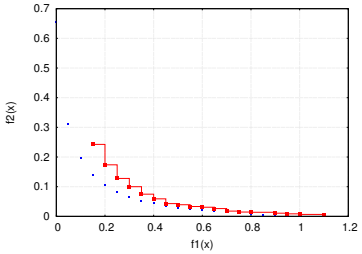
(b) ZDT2



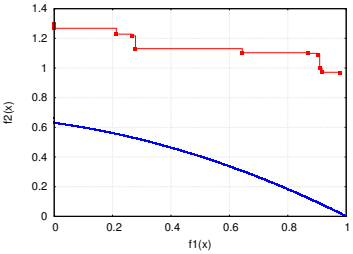
(c) ZDT3



(d) ZDT4



(e) ZDT5



(f) ZDT6

Figura 7.7: *Fronteiras de Pareto do otimizador básico no conjunto de testes ZDT. São apresentadas a fronteira das soluções não-dominadas e a fronteira ótima de cada problema de teste, para que possam ser comparadas.*

A figura 7.8 apresenta essa informação para cada problema de teste. Percebe-se que em praticamente todos os problemas de teste (com exceção do ZDT5),

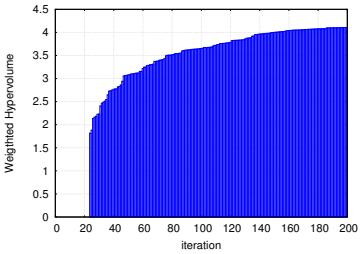
a área dominada (o hipervolume) é contabilizada apenas depois de algumas dezenas de iterações. Isso ocorre porque, antes desse limite, mesmo normalizada, o valor de pelo menos uma solução da fronteira de Pareto ainda correspondia a um ponto com coordenada superior a 2.1, acima do ponto de nadir, de modo que o valor do hipervolume é nulo.

7.4 Avaliação da Auto-Adaptação de Parâmetros Genéticos

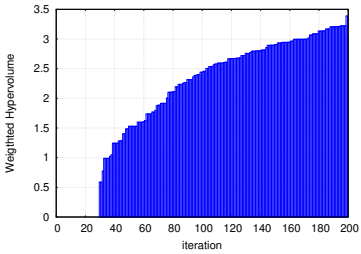
A auto-adaptação de parâmetros genéticos, representando o efeito de elementos epigenéticos e da pressão ambiental sobre elementos genéticos mais ou menos importantes permite retirar do projetista a responsabilidade de determinar esses parâmetros, de avaliar sobre cada gene ou códon a pressão de seleção e de identificar indiretamente os componentes e variáveis mais importantes no projeto de um sistema embarcado. Como ocorre em outros trabalhos, permitimos a auto-adaptação dos seguintes parâmetros genéticos: (1) Tamanho da população; (2) Quantidade de indivíduos-pais; (3) Quantidade de indivíduos-filhos; e (4) Taxa de aplicação de cada operador de variação genética.

Neste experimento avaliamos a eficiência da auto-adaptação de parâmetros dos operadores de variação, tanto em termos do indicador de hipervolume ponderado quanto em termos de convergência da otimização. O procedimento de execução deste experimento difere do procedimento geral apresentado na seção 7.1 pois a taxa de recombinação (φ) e a taxa de mutação (ζ) não correspondem aos valores empíricos fixados naquela descrição, mas evoluem ao longo das gerações, juntamente com as soluções do problema. Define-se assim que a recombinação e a mutação são os únicos operadores aplicados, ou seja, $\overrightarrow{\delta} = [\varphi, \zeta]$. Além de sua taxa de aplicação, evoluem também outros parâmetros dos operadores. No caso do operador de mutação, por exemplo, quando o códon codifica uma variável real (que é o caso dos experimentos neste capítulo), está associada ao operador (e a cada cromossomo, gene ou códon) uma distribuição de probabilidade (ξ) e, com ela, seus parâmetros (ver seção 3.3.4 e equação 3.8) . Tanto a distribuição quanto seus parâmetros podem evoluir com a solução.

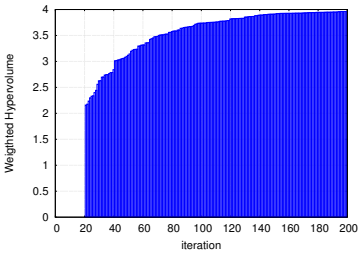
Desse modo, este experimento utiliza os mesmos parâmetros do algoritmo especificados na seção 7.1, ou seja, $N = 200$, $\alpha = 50$, $\mu = 20$, e $\lambda = 20$.



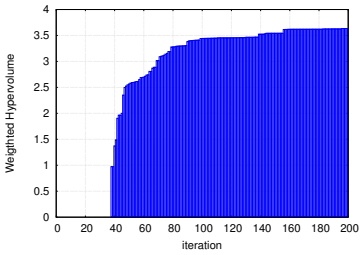
(a) ZDT1



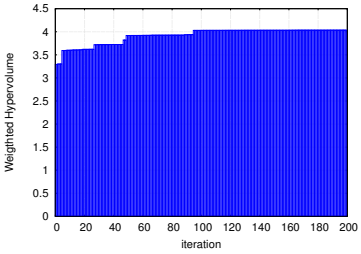
(b) ZDT2



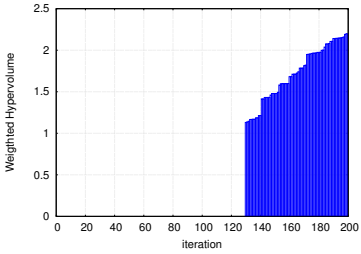
(c) ZDT3



(d) ZDT4



(e) ZDT5



(f) ZDT6

Figura 7.8: *Evolução do hipervolume do otimizador básico no conjunto de testes ZDT. A área preenchida corresponde ao hipervolume dominado pelas melhores soluções do otimizador a cada iteração, em relação ao ponto de referência, o que permite avaliar a convergência do algoritmo.*

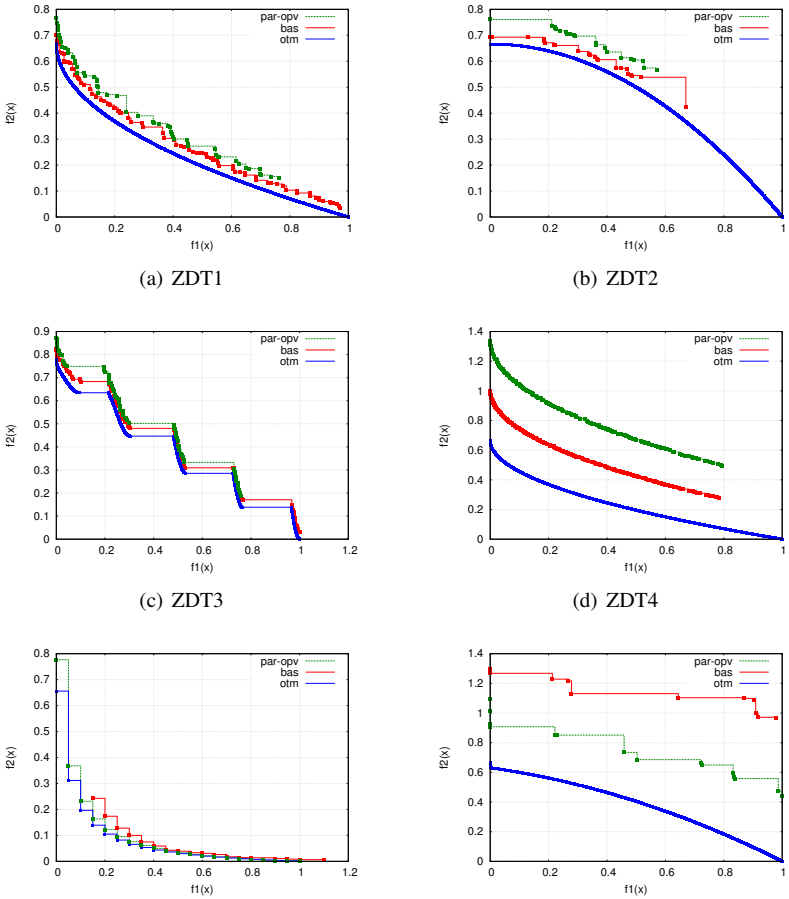
Todavia, define para cada gene do cromossomo uma região promotora com

parâmetros dos operadores de recombinação e mutação, que podem evoluir independentemente, e que são inicializados[3] com $\varphi = 1.0$ e $\zeta = 0.0333$. Para o operador de mutação, a distribuição inicialmente associada é uma polinomial, com parâmetro teórico $\eta_{mut} = 20$. O otimizador com auto-adaptação dos parâmetros dos operadores de variação passa a ser referenciado como $MOEA_{PAR\_OPV}$.

Os resultados deste experimento são apresentados nas figuras e tabelas seguintes. As fronteiras de Pareto para a versão com adaptações ($MOEA_{PAR\_OPV}$), para cada problema de teste, são apresentadas na figura 7.9, e são também comparadas com as fronteiras de Pareto ótimas. É possível verificar que a auto-adaptação individual dos parâmetros dos operadores de variação e de suas distribuições costuma levar a um conjunto não-dominado pior que a versão básica para a mesma quantidade de iterações, pois a convergência inicial demonstrou ser mais lenta. A auto-adaptação mostrou-se melhor para os problemas ZDT5 e ZDT6. Embora o problema ZDT4 demonstre ser aquele no qual a auto-adaptação teve os piores resultados, vale lembrar que esse é um problema com muitas fronteiras locais. Portanto, o que ocorreu é que a convergência mais lenta fez com que o conjunto de soluções ficasse preso numa fronteira local anterior. Em geral, assim que alguma solução "pula" uma fronteira local, o conjunto de soluções rapidamente converge à seguinte.

O valor final do indicador de hipervolume para cada um dos problemas de teste com o otimizador com auto-adaptação dos parâmetros $MOEA_{PAR\_OPV}$ é apresentado na tabela 7.3. Essa mesma tabela apresenta também sua relação com o indicador ótimo (da tabela 7.1) e com o indicador do otimizador básico, ou seja, $I^w_{HPAR\_OPV}/I^w_{HOTM}$ e $I^w_{HPAR\_OPV}/I^w_{HREF}$, sempre expressos em valores percentuais. Pelos resultados observa-se que, em média, a auto-adaptação dos parâmetros dos operadores genéticos foi 13,73% pior do que o algoritmo ótimo, e embora tenha tido desempenho pior que o algoritmo básico em vários problemas de teste, ainda foi 3,105% melhor, mas basicamente devido à grande melhora no problema ZDT6 (44,463%). Nesse tipo de problema, em que as soluções não estão bem distribuídas sobre a fronteira de Pareto, a variação diferenciada entre os genes pode realmente favorecer a busca em regiões específicas, fazendo com que os genes mais

[3]Esses valores poderiam ter sido inicializados aleatoriamente, mas optou-se por iniciá-los com os valores utilizados nos demais experimentos para melhor observar a evolução das taxas em cada gene.



(a) ZDT1 (b) ZDT2

(c) ZDT3 (d) ZDT4

(e) ZDT5 (f) ZDT6

Figura 7.9: *Fronteiras de Pareto do otimizador com auto-adaptação no conjunto de testes ZDT. São apresentadas a fronteira das soluções não-dominadas e a fronteira ótima de cada problema de teste, para que possam ser comparadas.*

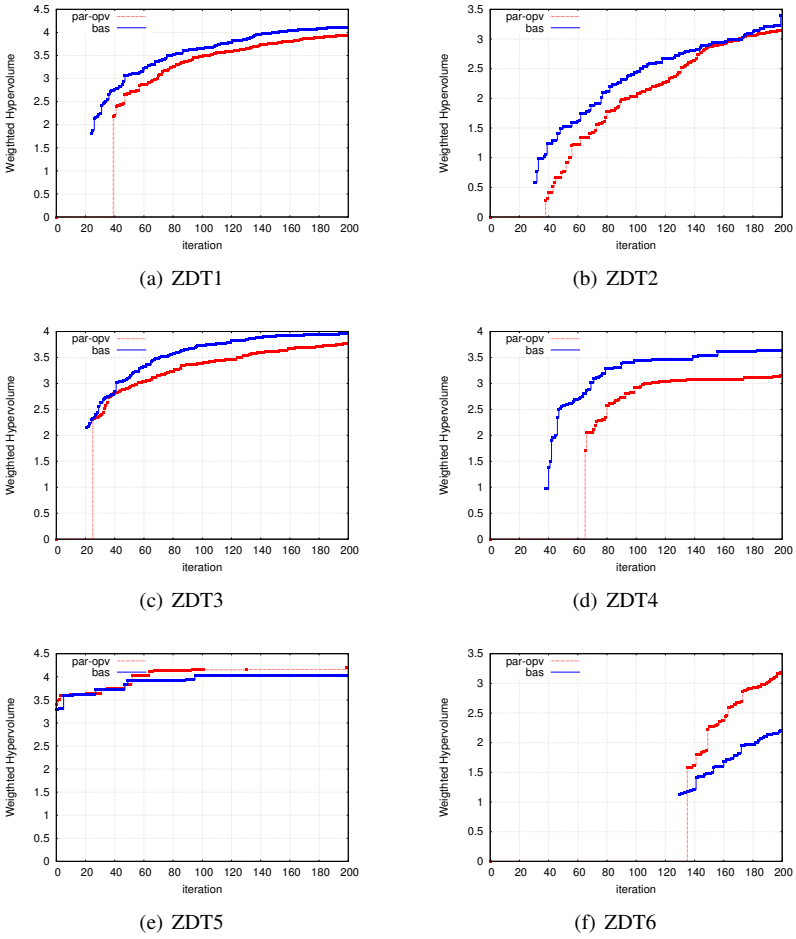
importantes tornem-se mais estáveis. Ignorando o problema ZDT6, a auto-adaptação de parâmetros usando regiões promotoras foi, em média, 5,17%

pior que o algoritmo básico. A figura 7.10 apresenta a evolução do indicador de hipervolume para o $MOEA_{PAR\_OPV}$ e para o $MOEA_{REF}$. Em todos os problemas de teste a área dominada pelo $MOEA_{PAR\_OPV}$ demora mais para aumentar, o que é esperado, já que os parâmetros de mutação e recombinação precisam se ajustar para cada gene. Entretanto, fica visível em alguns problemas de teste (ZDT1, ZDT2, ZDT3) que a área dominada aumenta pelas melhores soluções aumenta mais rapidamente no $MOEA_{PAR\_OPV}$ do que no $MOEA_{REF}$, após uma certa quantidade de iterações. Isso indica que, possivelmente, seu desempenho supere o do algoritmo básico para uma quantidade maior de iterações, o que compensaria o sobrecusto envolvido. Com isso, pode-se verificar que, ao permitir que cada gene possua uma taxa de mutação independente, diminui-se a taxa de convergência do algoritmo, que demora mais para atingir o ótimo.

Tabela 7.3: *Eficiência do otimizador com auto-adaptação com o benchmark ZDT. A eficiência é medida com o indicador de hipervolume ponderado, obtido a partir do conjunto de soluções não-dominadas para cada problema. A relação com o ótimo e com o otimizador básico também são apresentadas.*

	Problema de Teste					
	ZDT1	ZDT2	ZDT3	ZDT4	ZDT5	ZDT6
$I^w_{HPAR\_OPV}$	3,9330	3,1468	3,7625	3,1436	4,2003	3,1770
$\%\frac{I^w_{HPAR\_OPV}}{I^w_{HOTM}}$	93,923	79,355	93,281	75,071	97,255	78,739
$\%\frac{I^w_{HPAR\_OPV}}{I^w_{HREF}}$	95,801	92,828	94,976	86,508	104,05	144,46

Ressalta-se que a auto-adaptação dos parâmetros não foi guiada por nenhuma regra geral, como em [Zhang et al. 2009], o que realmente pode dificultar a convergência das soluções. Apesar dessa adaptação não ter trazido benefício em relação à eficiência da otimização (pelo menos não para poucas iterações), também não causou degradação significativa dos resultados, e a utilização de parâmetros diferenciados para cada elemento da estrutura genotípica e o uso de regiões promotoras traz ainda benefícios adicionais, pois é muito importante ao projeto de sistemas embarcados dirigidos à aplicação. Com elas é possível garantir que alguns componentes permaneçam estáveis, enquanto outros podem ser explorados, permitindo a exploração em níveis



(a) ZDT1 (b) ZDT2

(c) ZDT3 (d) ZDT4

(e) ZDT5 (f) ZDT6

Figura 7.10: *Evolução do hipervolume do otimizador com auto-adaptação no conjunto de testes ZDT. A área preenchida corresponde ao hipervolume dominado pelas melhores soluções do otimizador a cada iteração, em relação ao ponto de referência, o que permite avaliar a convergência do algoritmo.*

específicos e a definição de plataformas fixas, por exemplo. Além disso, é

necessário que genes correlacionados sejam todos (ou nenhum) passados aos indivíduos-filhos, de modo que é necessário desabilitar a recombinação entre esses genes.

7.5 Avaliação da Neutralidade Genética e Multiploidismo

O multiploidismo corresponde à existência de múltiplos genes alelos em cromossomos relacionados, sendo que o diploidismo é bastante comum entre organismos complexos em nossa biologia. O multiploidismo permite representar neutralidade genética (pois nem todos os genes são expressos) e competição entre genes, entre outras possibilidades. O multiploidismo permite que genes que estavam presentes em indivíduos bem adaptados (que se reproduziram) continuem presentes nos descendentes, mesmo que não expressos. Com isso, se determinada combinação de genes em certo indivíduo não for boa (devido à recombinação), torna-se mais fácil gerar novas combinações entre os genes antigos se eles forem preservados no genoma (como genes silenciosos, recessivos ou lixo genético), do que gerá-los novamente por mutação.

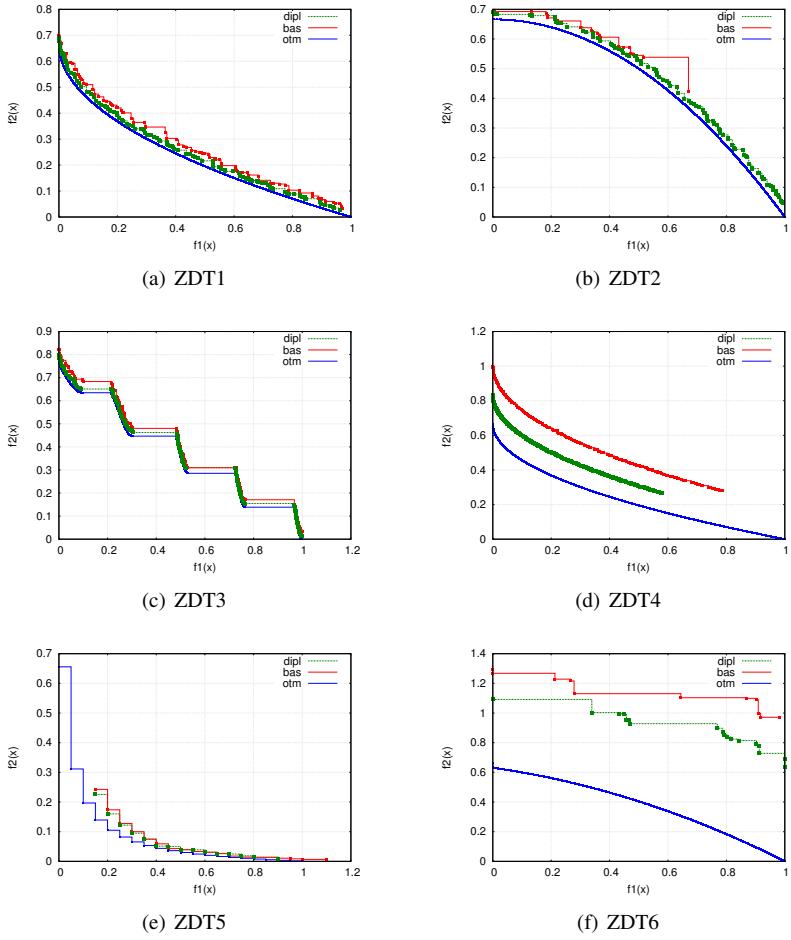
Neste experimento avaliamos a eficiência do diploidismo e da neutralidade genética por ele gerada. O procedimento de execução deste experimento é idêntico ao procedimento geral apresentado na seção 7.1, tendo sido alterada apenas a estrutura genotípica dos indivíduos que representam as soluções. Cada indivíduo é representado por um cromossomo com dois genes por locus, por seja, dois cromossomos homólogos. A abordagem utilizada neste experimento usa apenas genes dominantes ou recessivos, embora a codominância também tenha sido desenvolvida. Cada gene possui uma região promotora que informa sua "dominância", representada por um número inteiro. Um gene é dominante em relação ao seu alelo se esse número inteiro é maior, e apenas o valor do gene dominante é considerado na tradução do genótipo em fenótipo (o gene recessivo é neutro). A dominância de todos os genes é atualizada no final de cada iteração, após a etapa de reprodução, quando os melhores indivíduos são selecionados e colocados no conjunto elitista de melhores soluções. Nesse momento, todos os indivíduos da população são atualizados. Os indivíduos do conjunto elitista de melhores soluções (ou seja, indivíduos não Pareto-dominados por outros) têm a "dominância" de todos

seus genes dominantes incrementada, e os indivíduos da população que não pertencem ao conjunto de melhores soluções têm a "dominância" de seus genes dominantes decrementada. Nada ocorre com os genes recessivos, embora essa alteração de valores possa alterar qual dos genes é recessivo ou dominante. Essa abordagem reforça a dominância dos genes alelos que, em conjunto com os demais genes dominantes, geram soluções ótimas, e enfraquece a dominância de genes alelos que, em conjunto com os demais, não geram soluções tão boas. O otimizador com diploidismo passa a ser referenciado como $MOEA_{DIPL}$.

Os resultados deste experimento são apresentados nas figuras e tabelas seguintes. As fronteiras de Pareto para a versão com diploidismo ($MOEA_{DIPL}$), para cada problema de teste, são apresentadas na figura 7.11, e são também comparadas com as fronteiras de Pareto da versão básica do algoritmo e com a fronteira ótima, como nos experimentos anteriores. É possível verificar que o diploidismo leva a um conjunto não-dominado melhor que a versão básica para a mesma quantidade de iterações. Sua convergência também é mais rápida, como é visível na figura 7.12, que mostra a evolução do indicador de hipervolume. A eficiência do $MOEA_{DIPL}$ foi, em média, 6,18% melhor que o otimizador básico, sem adaptações. Uma das maiores melhoras em relação ao algoritmo básico ocorreu na fronteira não-convexa do problema ZDT2, que ficou mais próxima e melhor distribuída, com uma melhora de 12,73%. No problema ZDT4, o $MOEA_{DIPL}$ conseguiu avançar mais fronteiras locais que o algoritmo básico, porém não teve tempo de explorar toda essa nova fronteira, gerando soluções bem distribuídas, de modo que a área dominada melhorou apenas 0,96%.

7.6 Discussão

Os resultados das avaliações feitas nos experimentos apresentados nas seções anteriores são sintetizados na figura 7.13. Nessa figura estão representados o valor do indicador de hipervolume ponderado para os algoritmos $MOEA_{OTM}$,$MOEA_{REF}$,$MOEA_{PAR\_OPV}$ e $MOEA_{DIPL}$, já normalizados em relação ao $MOEA_{REF}$ para cada problema de teste, ou seja $I^w_{HREF} = 1$. Nessa figura é importante analisar a proximidade dos valores $MOEA_{PAR\_OPV}$ e $MOEA_{DIPL}$ em relação a $MOEA_{OTM}$,$MOEA_{REF}$. Percebe-se, então, que é



(a) ZDT1 (b) ZDT2

(c) ZDT3 (d) ZDT4

(e) ZDT5 (f) ZDT6

Figura 7.11: *Fronteiras de Pareto do otimizador com diploidismo no conjunto de testes ZDT. São apresentadas a fronteira das soluções não-dominadas e a fronteira ótima de cada problema de teste, para que possam ser comparadas.*

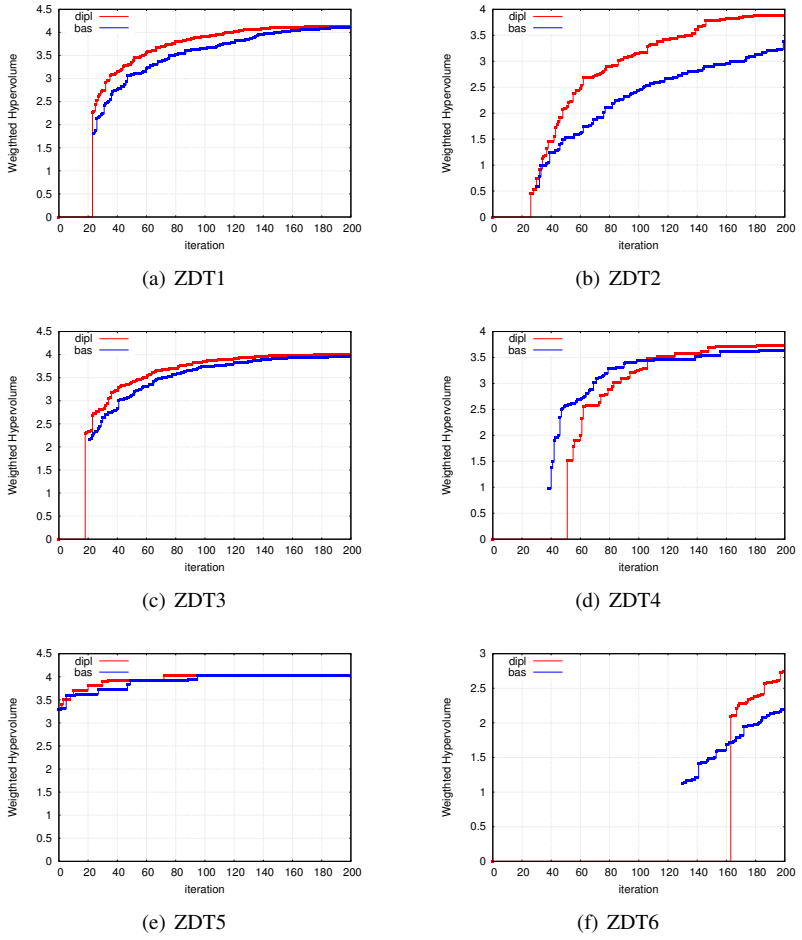
nos problemas ZDT2, ZDT4 e ZDT6 que a inclusão do multiploidismo mais se destacou em relação ao algoritmo básico, ou seja, essa adaptação traz me-

Tabela 7.4: *Eficiência do otimizador com diploidismo com o benchmark ZDT. A eficiência é medida com o indicador de hipervolume ponderado, obtido a partir do conjunto de soluções não-dominadas para cada problema. A relação com o ótimo e com o otimizador básico também são apresentadas.*

	Problema de Teste					
	ZDT1	ZDT2	ZDT3	ZDT4	ZDT5	ZDT6
I^w_{HDIPL}	4,1348	3,8846	4,0001	3,7397	4,0350	2,7454
$\%\frac{I^w_{HDIPL}}{I^w_{HOTM}}$	98,742	97,958	99,170	89,304	93,428	68,044
$\%\frac{I^w_{HDIPL}}{I^w_{HREF}}$	100,72	114,59	100,97	102,91	99,959	124,84

lhores resultados em problemas com a fronteira de Pareto não-convexa, ou em que há mínimos locais ou então em que as soluções estão concentradas em certa região da fronteira de Pareto.Contudo, não apresentou resultados são bons no ZDT5, em que o problema é deceptivo e a codificação foi uma string de bits. O uso da região promotora para auto-adaptação dos parâmetros dos operadores de mutação e recombinação demonstrou retardar a convergência do algoritmo, o que exige mais avaliações das soluções (mais iterações) para alcançar boas soluções. A exceção foi o problema ZDT6, com soluções concentradas, possivelmente porque mesmo com pouca variação genética, as soluções concentradas foram encontradas rapidamente.

Os experimentos apresentados neste capítulo mostram, de forma empírica, que as adaptações desenvolvidas trazem alguma melhora em relação ao desempenho da otimização, pelo menos para algumas categorias de problemas. Certamente, essas adaptações implicam um aumento do sobrecusto no modelo, em termos de memória e tempo de processamento, principalmente. Esse sobrecusto não foi avaliado nesta tese, pois demandaria implementações totalmente distintas e a exclusão de alguns elementos utilizados nos experimentos, como os *dataloggers* que coletam informações para análise dos resultados, que não seriam necessários num otimizador final, mas que também afetam o sobrecusto. A melhora de desempenho obtida nos conjuntos de funções de teste normalmente utilizados como *benchmarks* na literatura, como o ZDT, utilizado nesta tese, pode realmente não compensar seu sobrecusto. Porém, em problemas mais complexos e que não podem ser (tão



(a) ZDT1 (b) ZDT2

(c) ZDT3 (d) ZDT4

(e) ZDT5 (f) ZDT6

Figura 7.12: *Evolução do hipervolume do otimizador com diploidismo no conjunto de testes ZDT. A área preenchida corresponde ao hipervolume dominado pelas melhores soluções do otimizador a cada iteração, em relação ao ponto de referência, o que permite avaliar a convergência do algoritmo.*

facilmente) expressos em termos que equações matemáticas, como é o caso

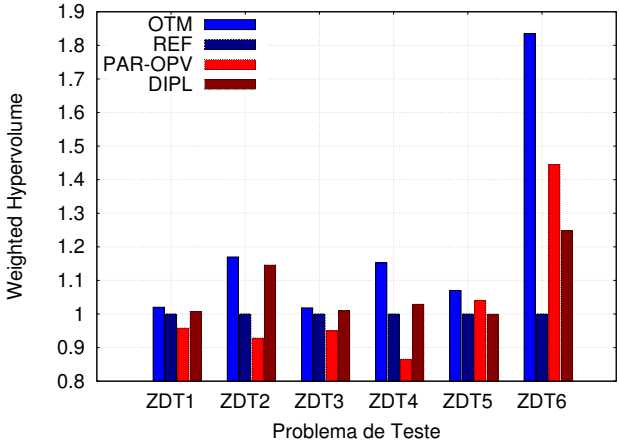


Figura 7.13: *Comparativo do hipervolume para os experimentos realizados. São apresentados o indicador de hipervolume ponderado para os algoritmos $MOEA_{OTM}$, $MOEA_{REF}$, $MOEA_{PAR\_OPV}$ e $MOEA_{DIPL}$, para cada problema de teste utilizado.*

do projeto de sistemas embarcados no nível de granularidade que se deseja operar na ADESD, isso pode ser diferente. Nesses casos, a avaliação de cada solução é complexa e demorada, e a melhora no desempenho da otimização, reduzindo a quantidade de iterações, pode compensar bastante o tempo de avaliação. Além disso, as adaptações desenvolvidas parecem indispensáveis nesse tipo de problema, no qual é necessário controlar o comportamento de cromossomos, genes e códons específicos, por exemplo.

PARTE IV

Conclusões

CAPÍTULO 8

CONCLUSÕES

Somos ótimos para detectar as falhas dos outros, mas míopes para enxergar as nossas.
—AUGUSTO JORGE CURY

Se não estivermos dispostos a ajudar uma pessoa a vencer as suas falhas, há pouco valor em apontá-las.
—ROBERT J. HASTINGS

Este é o capítulo final da parte textual desta tese, e apresenta as conclusões finais. Ele é dividido em 3 seções. A seção 8.1 relembra rapidamente o contexto desta tese, os problemas tratados e seus objetivos principais, e mostra como os objetivos foram alcançados. A seção 8.2 condensa as contribuições principais da tese, suas vantagens e benefícios, mas principalmente explicita seu escopo, suas limitações e os problemas ainda existentes. Por fim, a seção 8.3 explora os problemas remanescentes, tanto aqueles não resolvidos quanto aqueles gerados a partir da solução desenvolvida, e propõe temas para futuras pesquisas e possíveis estratégias a serem utilizadas.

8.1 Contextualização e Considerações

O projeto de sistemas embarcados tem se tornado mais complexo à medida que avançam a tecnologia e as aplicações, o que exige o aperfeiçoamento contínuo de metodologias e técnicas de projeto. Em praticamente todas as metodologias modernas, a exploração do espaço de projeto tem se tornado uma etapa fundamental, devido à sua importância na qualidade da solução final. De modo geral, essa exploração corresponde a uma otimização multiobjetivo, e a utilização de algoritmos evolucionários para a exploração do espaço de projeto em sistemas embarcados tem sido alvo de pesquisas há mais de uma década. Apesar disso, vários problemas persistem. Dentre eles, os problemas tratados nesta tese são a representação diferenciada de sistemas embarcados e de seus componentes[1] e configurações, incluindo o software

[1] assume-se que o projeto de sistemas embarcados baseia-se em componentes, conforme a metodologia de projeto dirigido pela aplicação (ADESD).

embarcado (que normalmente é subestimado), o hardware físico e também sintetizável, demais informações que permitam a exploração em diferentes níveis hierárquicos e a inclusão de adaptações no modelo evolucionário que suporta tal representação diferenciada e sobre o qual é realizada a otimização multiobjetivo.

Nesta tese foram descritos os metamodelos desenvolvidos e que representam (1) o projeto de um sistema embarcado, que armazena informações sobre os requisitos e especificações de entrada e também sobre os resultados e saídas do mesmo, e que gera uma base de conhecimento que permite que futuras explorações possam beneficiar-se de boas soluções já encontradas; (2) características de sistemas embarcados que podem ser usadas como critérios de otimização e exploração, que armazena estimativas realizadas e o contexto no qual foram obtidas, o que permite a evolução do modelo usado para estimá-las e também que objetivos diferentes do usuais (área, memória, energia, latência) possam ser explorados; (3) componentes que compõem os sistemas embarcados e todas as características exigidas pela ADESD para sua configuração e adaptação e geração automática dos sistemas que eles compõem, bem como maior nível de detalhamento nos componentes que definem o suporte de hardware; e (4) outros elementos associados aos componentes, como agrupamentos de componentes que formam um "*template*" e as ferramentas de software associadas a arquiteturas e PLDs e realizam a transformação dos componentes, tipicamente compilação e síntese.

Para tornar possível a exploração do espaço de projeto em sistemas embarcados cujo projeto e seus componentes foram representados pelos metamodelos criados, foi desenvolvido também um modelo evolucionário para exploração do espaço de projeto (M3EP), que mapeia as informações dos componentes e de sua comunicação que precisam ser explorados (*traits*, características configuráveis, dependências alternativas e mapeamentos) ao genótipo de um indivíduo, no contexto dos algoritmos evolucionários, e é sobre esse modelo (M3EP) que se realiza a otimização multiobjetivo.

Para que o modelo evolucionário represente adequadamente todas as informações necessárias, foram propostas adaptações aos algoritmos evolucionários, incluindo a representação de elementos de inspiração biológica e bioquímica que eram ignorados ou não devidamente adequados nas demais abordagens encontradas na literatura,e que incluem (1) Auto-adaptação de parâmetros genéticos, que permite que parâmetros evoluam juntamente com

as soluções do problema, bem como permite diferentes parâmetros dos operadores para cada indivíduo, cada cromossomo, cada gene ou cada códon, e que variem conforme condições específicas do problema, como a geração atual, a diversidade da população, ou outras condições epigenéticas, utilizando uma nova estrutura denominada "região promotora"; (2) Tratamento da epistasia, que permite que variáveis correlacionadas possam ser mantidas juntas nas soluções; (3) Neutralidade genética, que mantém inativos elementos (códons, genes, cromossomos) que foram parte de boas soluções em gerações anteriores, de modo que podem ser mais facilmente retomadas do que se fossem simplesmente eliminadas; (4) Multiploidismo, que corresponde a uma forma de neutralidade de genes e que permite que genes alelos participem do fenótipo apenas na medida em que melhoram a qualidade da solução final.

Assim, demonstrou-se, empiricamente, que o M3EP é capaz de representar um sistema embarcado projetado pela ADESD, incluindo seus componentes, configurações e demais informações passíveis de exploração, a comunicação entre esses componentes e numa estrutura que o representa desde o nível de sistema até o nível de arquitetura, permitindo a exploração multihierárquica. Contudo, não foram apresentados os resultados da exploração do espaço de projeto em sistemas embarcados reais usando esse modelo evolucionário. Devido à inexistência de equações que descrevem as funções-objetivo específicas para o sistema sendo projetado e o consequente desconhecimento de sua fronteira de Pareto, a avaliação formal da qualidade das soluções geradas pela exploração do espaço de projeto de sistemas embarcados é impraticável. Além disso, a exploração com o M3EP não gera sistemas embarcados completamente funcionais, devido à complexidade de evoluir suportes de hardware (circuitos eletrônicos), o que exige do projetista bons conhecimentos de eletrônica e bastante tempo para que cada sistema projetado seja efetivamente concluído.

Deste modo, nesta tese, a avaliação da qualidade das soluções foi realizada submetendo cada nova adaptação incluída no modelo evolucionário a um conjunto de problemas de testes conhecido na literatura, e que submetem o modelo evolucionário e o algoritmo genético a diferentes obstáculos, permitindo verificar sua capacidade de produzir boas soluções (próximas da fronteira de Pareto e bem distribuídas) em diferentes condições. Utilizando o indicador de qualidade hipervolume ponderado para avaliar formalmente a qualidade das soluções pode-se, indiretamente, estimar o quão boas seriam as soluções encontradas na otimização de problemas reais. Considero essa

uma avaliação importante e necessária antes da utilização do modelo evolucionário desenvolvido para a exploração em sistemas embarcados reais. Assim, foram descritos e realizados experimentos que avaliaram o desempenho na otimização das adaptações de inspiração biológica que compõem o modelo evolucionário para exploração do espaço de projeto.

Portanto, foi cumprido o objetivo de desenvolver um modelo de otimização multiobjetivo baseado em algoritmos evolucionários e que permita a exploração do espaço de projeto de sistemas embarcados dirigidos pela aplicação, com as características especificadas: representação de componentes configuráveis e adaptáveis, adaptação do suporte hardware, tanto físico quanto sintetizável e exploração em diferentes níveis. Também foi cumprido o objetivo específico de avaliar as novas estruturas e operadores de inspiração biológica desenvolvidos para o modelo evolucionário.

Como parte dos resultados parciais desta tese foram publicados 14 artigos científicos, sendo 1 (um) em periódico B2 (JOT) [Schulter et al. 2007], 3 (três) em eventos B2 (ETFA, INDIN) [Santos et al. 2006, Santos et al. 2006, Cancian, Fröhlich e Stemmer 2007], 4 (quatro) em eventos B5 (WTR, WSO) [Cancian, Stemmer e Fröhlich 2006, Santos, Cancian e Fröhlich 2006, Marcondes et al. 2006, Cancian et al. 2007], e 6 (seis) em eventos sem qualificação (I2TS, EPS, RTSS-WIP, etc) [Cancian, Stemmer e Fröhlich 2006, Pereira e Cancian 2008, Marcondes et al. 2009, Marcondes et al. 2009, Cancian, Stemmer e Fröhlich 2009, Cancian, Stemmer e Fröhlich 2009]. Alguns dos eventos no qual houve publicação, como ETFA, INDIN e I2TS, por exemplo, tinham, no momento da publicação, classificação superior à apresentada. Além disso, os resultados mais recentes da tese estão sendo submetidos a eventos científicos, mas ainda não foram publicados, de modo que espera-se produzir ainda algumas publicações em eventos com classificação não inferior a B1.

8.2 Contribuições e Limitações

Dentre as primeiras contribuições desta tese, pode-se citar a avaliação da eficiência de otimização de novos elementos nos algoritmos evolucionários para otimização multiobjetivo, usando um conjunto de problema de teste

e um indicador de qualidade consagrados. Foram incluídos elementos que representam o multiploidismo, presente em praticamente qualquer organismo complexo de nossa biologia, e que permite a neutralidade genética e a competição entre genes, entre outras características. Não tenho conhecimento de outras pesquisas que representam esses elementos. A inclusão desses elementos demonstrou aumentar a eficiência em termos de otimização para os problemas de teste e pode ser útil no projeto de sistemas embarcados ao manter nas soluções atuais, mesmo que "silenciados" ou neutros, os componentes e configurações que foram bons em soluções passadas.

Outra contribuição em relação aos elementos incluídos no modelo evolucionário foi a inclusão do códon, que pode representar diferentes tipos de variáveis presentes em problemas reais mais complexos, além das tradicionais strings de bits e variáveis reais. Não tenho conhecimento de outras pesquisas que representem esse elemento e permitam a existência de genes heterogêneos. Ao incluir mais um nível de informação dentro do gene, foi possível criar uma estrutura única de um elemento atômico e mais complexo, que pode representar um componente de software ou hardware, por exemplo, com todas suas configurações, parâmetros, dependências estruturais e mapeamentos.

A auto-adaptação diferenciada de parâmetros já existe há alguns anos, mas nas soluções existentes ela é realizada apenas em nível de iterações ou de indivíduos. Contudo, em problemas mais complexos pode ser necessário criar comportamentos específicos para cada indivíduo, cromossomo, gene ou códon, como ocorre na natureza. Nesta tese utilizei o conceito biológico de região promotora para criar uma estrutura que permite alterar o comportamento de qualquer elemento genético separadamente. Demonstrei sua eficiência na representação da dominância e competição genética, bem como na auto-adaptação de parâmetros dos operadores genéticos de variação (mutação e recombinação). Porém, ela tem várias outras utilidades que podem ser exploradas. No contexto de sistemas embarcados, ela poderia ser aplicada para possibilitar a exploração em níveis específicos, ou fazer exploração em diversos níveis, criar "templates" de componentes e de micro-arquiteturas, entre muitos exemplos possíveis.

Muitas contribuições em relação à exploração do espaço de projeto em sistemas ficaram obscurecidas pela impossibilidade prática de comprovação formal da qualidade das soluções geradas, o que também levou à supressão,

no texto da tese, de vários modelos, desenvolvimentos e alguns experimentos de exploração do espaço de projeto que não podiam ter sua qualidade comprovada. Contudo, mesmo sem experimentos comprobatórios, tentei explicitar, ao longo do texto, a aplicabilidade e utilidade da abordagem desenvolvida no projeto de sistemas embarcados dirigidos pela aplicação. Uma contribuição nesse aspecto é a representação do software embarcado como componentes de granularidade fina (mais fina do que a encontrada em outras soluções), separando a aplicação embarcada do seu suporte de execução (que costuma ser ignorado), o que permite maior exploração do software, tanto com a aplicação de parâmetros de configuração (*traits*) quanto a aplicação de aspectos e demais informações que permitem de forma automática sua seleção, configuração, adaptação, e geração da imagem final (binário executável).

Outra contribuição consiste na exploração de novos suportes de hardware, tanto físico quanto sintetizável, alterando as conexões entre os componentes de hardware, e não apenas a parametrização dessas plataformas, e permite a evolução do hardware em nível de componentes e não de portas-lógicas, como nas abordagens atuais. O modelo desenvolvido também permite que alguns suportes de hardware ou "pedaços" desses suportes de hardware, como micro-arquiteturas ou sub-circuitos, sejam fixados (usando as regiões promotoras para impedir sua variação), ou que sua estrutura de conexão seja fixada, mas que os componentes sejam parametrizáveis, o que aumenta as possibilidades de exploração, desde um extremo em que qualquer coisa no hardware (em nível de componentes pré-manufaturados) pode ser alterado até outro extremo em que os suportes de hardware são fixos. Embora o desenvolvimento realizado não produza circuitos funcionais, ainda assim pode pode ser mais produtivo do que começar a partir do nada. Não tenho conhecimento de outras pesquisas que realizem a exploração tanto do suporte de hardware físico quanto sintetizável, e sua evolução.

O projetista também pode expandir o modelo dos componentes, incluindo novos atributos a tipos específicos de componentes, e então utilizar esses novos atributos na exploração do espaço de projeto, através do metamodelo de caracterização dos custos do sistema, que evolui um modelo simbólico (GEP) de obtenção de estimativas para qualquer atributo/característica representado nos componentes. Com isso, a exploração não fica limitada à otimização de apenas dois ou três objetivos típicos, como área, memória e energia, por exemplo. Qualquer característica dos componentes do sistema, como preço, peso ou tamanho físico, por exemplo, pode ser usado como

critério de otimização. Isso é possível, em parte, apenas porque permite-se que o projetista forneça "feedback" dessas características em sistemas projetados, realimentando o metamodelo de caracterização de custos do sistema, que pode evoluir um modelo de estimativas dessas novas características (baseado na evolução de equações matemáticas por GEP) e assim melhorar gradativamente a qualidade das explorações futuras.

Por fim, também é uma contribuição a representação do sistema embarcado em vários níveis, desde a conexão física de componentes ao mapeamento de componentes de software a arquiteturas, num único MOEA, e que utiliza informações de projetos passados relacionados, o que permite que as melhores soluções de outros projetos sejam incorporadas à população inicial de novos projetos.

Entretanto, esta tese também possui diversas limitações à sua aplicabilidade e capacidade de exploração do espaço de projeto em sistemas embarcados. Entre as várias limitações desta tese, podemos citar que não é feita a geração automática e funcional de novos suportes de hardware, pois não foram implementadas regras suficientes que especifiquem a adaptação (no sentido evolucionário) de novos circuitos, nem houve a integração de simuladores de circuitos eletrônicos, que guiariam o processo de otimização. Também não há prova formal que se forem implementadas tais regras, garante-se que a otimização será capaz de inferir todos os componentes eletrônicos e de configurá-los de forma que circuitos eletrônicos complexos funcionais sejam gerados automaticamente. Realmente não parece ser esse o caso, e a potencialidade dessa exploração ainda precisa ser investigada, pois não fez parte do escopo desta tese. Porém, mesmo com essas limitações e mesmo incipiente, a evolução de suportes de hardware fornece uma versão preliminar da plataforma para os projetistas de hardware, o que pode ser útil.

A abordagem utilizada para exploração faz a alocação de recursos de software e de hardware e também o mapeamento de software a recursos de hardware, e ainda mais, como o mapeamento de hardware sintetizável a hardware físico e a alocação de requisitos de software (aplicativo) a recursos também de software (de suporte). Porém, não faz o escalonamento. Não foram desenvolvidos os mecanismos necessários para determinar os instantes de início e de término de execução dos componentes de software, o que não é possível sem a adição de novos modelos. Isso porque, na abordagem utilizada, o software não é modelado como um grafo de precedências de tarefas com

anotação do tempo de execução de cada uma. Não considerei que essa seria uma abordagem condizente com a ADESD, e não foi implementada. Porém, também não propus nenhuma solução alternativa, de modo que esse ainda é um problema a ser tratado.

O tratamento das penalidades aplicadas às soluções inviáveis no modelo evolucionário de exploração do espaço de projeto (M3EP) considera apenas os erros listados pelas ferramentas de geração de sistemas, o que é uma abordagem muito básica. Embora essa seja apenas uma questão de implementação, e não de restrição do modelo em si, a qualidade das soluções (sistemas embarcados) pode ser melhorada consideravelmente se o avaliador de fenótipo desse problema for melhor explorado, refinando o procedimento de tratamento de penalidade das soluções inviáveis e também de correção automática dessas soluções.

8.3 Pesquisas Futuras

As limitações impostas pela versão atual da solução desenvolvida, e mesmo suas contribuições e os problemas solucionados, abrem caminho para novas pesquisas e desenvolvimentos em diferentes tópicos dos temas abordados. Inicialmente, em relação aos algoritmos evolucionários, dentre as possíveis pesquisas futuras, pode-se citar uma abordagem diferente de codificação e de tradução. A solução adotada, bem como a maioria das demais soluções disponíveis, preferiu uma codificação das variáveis do problema mais complexa e próxima das variáveis do problema em si, codificando variáveis reais como números reais nos genes, por exemplo. Com isso, a tradução do genótipo para o fenótipo é praticamente direta. Embora essa abordagem tenha vantagens computacionais e de desempenho obvias, ela é contrária ao que ocorre na natureza, em que a codificação é muito diferente das variáveis do problema (que correspondem às características fenotípicas dos organismos vivos). Não se codifica diretamente as variáveis, mas instruções para chegar às variáveis, e o processo de tradução é que é complexo e com várias etapas (transcrição, tradução, estruturação de proteínas, redes metabólicas, etc). A avaliação do desempenho de otimização colocando a ênfase da otimização no processo de tradução e não de codificação, pode ser um tema de pesquisa interessante.

Em relação à otimização multiobjetivo, pesquisas futuras podem incluir uma abordagem iniciada recentemente, em que um indivíduo de um algoritmo evolucionário não representa uma única solução ao problema e a população de uma geração representa um conjunto de soluções. Dado que estamos interessados em obter aproximações para o conjunto de soluções ótimas e comparar conjuntos de soluções, um único indivíduo poderia representar um conjunto de soluções e todas as estratégias de otimização passam a operar sobre conjuntos de soluções, e não mais sobre soluções individuais. Vários tópicos específicos nessa abordagem ainda são temas relevantes de pesquisa.

Outra questão que surge com a análise de diferentes heurísticas e parâmetros é a otimização das escolhas dos diferentes componentes de otimização para um problema específico. Os otimizadores incluem parâmetros que podem ser auto-adaptados (ou seja, otimizados), como foi feito nesta tese e em outras pesquisas encontradas na literatura, o que libera o projetista da tarefa de especificar, empiricamente, o tamanho da população, a quantidade de indivíduos-pai e de indivíduos-filhos, e as taxas de mutação e recombinação, por exemplo. Porém, o projetista ainda especifica empiricamente que vai utilizar apenas os operadores de mutação simples e de recombinação de n-pontos (e não outros tipos desses operadores ou mesmo outros operadores), que utilizará a estratégia de seleção NSGA2 (e não SPEA2, D-MOEA ou outra qualquer), que utilizará a dominância de Pareto (e não r-dominância, α-dominância ou outra qualquer), que utilizará certa estratégia de elitismo, de garantia de diversidade, e assim por diante. O que ocorre é que o projetista ainda especifica empiricamente esses componentes/estratégias da otimização, ou então escolhe empiricamente algumas combinações, compara-as e escolhe a melhor. Contudo, esse é um processo empírico de tentativa e erro, e não um processo de otimização. Um tema interessante pode ser um meta-otimizador, ou seja, um otimizador no qual o indivíduo codifica todos esses componentes/estratégias de um otimizador. A tradução do genótipo de um desses indivíduos deve obter essas estratégias e a avaliação da aptidão de um desses indivíduos significa fazer uma otimização completa do problema com um otimizador com essas estratégias e a obtenção de um indicador de desempenho, como o hipervolume, por exemplo. Assim, o indivíduo que codificar melhores componentes/estratégias de otimização deve ser preservado na população e evoluir para definir as melhores estratégias de otimização para um problema específico. Vários desdobramentos dessa abordagem parecem inte-

ressantes.

Em relação aos sistemas embarcados e a exploração de seu espaço de projeto, dentre as possíveis pesquisas futuras, podemos citar, inicialmente, aperfeiçoamentos técnicos e complementações na implementação da solução proposta e que devem permitir obter resultados mais relevantes e abrir outros caminhos. Dentre os aperfeiçoamentos técnicos que podem ser feitos, estão (1) A inclusão de simuladores físicos, como o SPICE ou que utilizem o SPICE (como o LabCenter/Proteus/ISIS) para a evolução de suportes de hardware físico; (2) A melhoria das regras de interconexão de componentes de hardware e de comunicação de componentes de software entre nodos distintos; (3) A inclusão de co-simuladores, modelos analíticos ou outros estimadores para avaliar a aptidão das soluções em menos tempo; (4) A implementação de estratégias que permitam escolher o melhor estimador para uma solução (com base na geração da população, viabilidade da solução, ou outras soluções dominadas, por exemplo); ou ainda (5) o desenvolvimento e cadastro de novos componentes, incluindo componentes descritos em SystemC ou como máquinas de estados, o que aumentaria a quantidade de soluções alternativas. Por fim, é necessário o projeto e geração final de uma boa quantidade de sistemas embarcados reais e funcionais, utilizando a solução desenvolvida, para avaliá-la de forma direta. A avaliação dos sistemas gerados pode levantar novas questões de pesquisa.

Pesquisas futuras também podem incluir um estudo sobre a evolução das equações de funções-objetivos desconhecidas que envolvam características dos componentes utilizados na solução, como proposto e desenvolvido com o metamodelo de caracterização dos custos do sistema. A efetividade dessa abordagem apenas poderá ser confirmada adequadamente com a geração de sistemas finais e a realimentação dos valores de métricas desses sistemas pelo projetista. Além disso, imagina-se que questões dinâmicas dos sistemas, como interação com usuário e com o ambiente, por exemplo, podem ter forte impacto nas métricas dos sistemas reais, e se essas questões podem ser capturadas adequadamente por equações que evoluem, ainda é uma questão a ser pesquisada.

PARTE V

Apêndices

APÊNDICE A

SUITE DE FERRAMENTAS DE SUPORTE DA ADESD

Matemática é fácil; design é difícil.
—JEFFREY VEEN

Este apêndice apresenta rapidamente a nova suite de ferramentas da ADESD, fornecendo uma descrição sucinta de sua interface e de sua utilização, com o único propósito de fornecer um guia superficial de sua utilização e também de explicitar como os modelos e metamodelos desenvolvidos aparecem ao projetista de sistemas embarcados e de como a infraestrutura de otimização e dos algoritmos evolucionários fica transparente a ele.

A.1 Visão Geral

A suite de ferramentas foi desenvolvida em linguagem Java 6, utilizando a IDE Netbeans. Sua tela inicial é apresenta na figura A.1. Nela são visíveis as janelas de console do sistema e de log do sistema, e também a estrutura do menu principal, que possui as seguintes opções:

- System. Opções gerais de configuração da suite de ferramentas, realização de testes de configuração do ambiente (estrutura de diretórios criada, ferramentas instaladas, etc), mudança da decoração dos elementos gráficos (*look and feel*) e de saída do sistema;
- Repository. Geração semi-automática do repositório de componentes a partir de modelo de domínio, gerenciamento do repositório (novo, abrir, salvar, fechar, editar);
- Design system. Gerenciamento de projeto de sistema embarcado (novo, abrir, salvar, fechar, editar), e também fornecer *feedback* sobre características dos custos do sistema projetado;
- Application. Gerenciamento das aplicações-alvo (abrir, fechar), e também análise das aplicações, com a extração de seus requisitos (invocação da ferramenta `Analyser`);

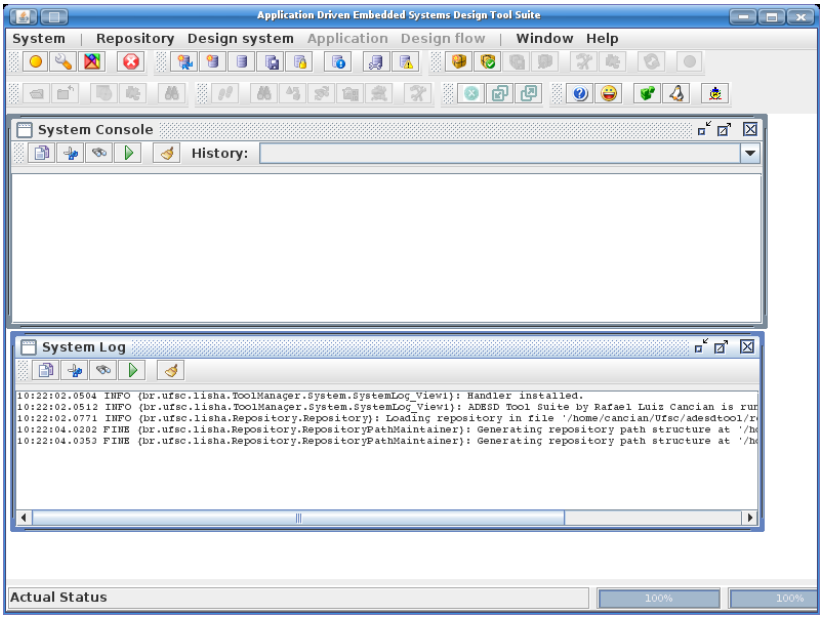


Figura A.1: *Interface principal da suite de ferramentas*

- Design flow. Configuração e execução de cada etapa do fluxo de projeto de um sistema embarcado: análise, configuração, exploração, simulação e geração do sistema final;
- Window. Funções básicas de acesso a (e organização de) janelas específicas da suite de ferramentas.
- Help. Opções de ajuda e documentação.

De forma geral, o projetista pode usar a suite de ferramentas com uma navegação sequencial dos menus apresentados, pois os passos normalmente realizados num projeto de sistema embarcado são:

- Verificar a instalação das ferramentas associadas e configurar a suite de ferramentas;

- Abrir um repositório de componentes com a infraestrutura necessária ao projeto;
- Definir as informações do sistema sendo projetado, como as restrições de projeto e regiões de interesse, por exemplo;
- Abrir as aplicações-alvo do projeto, previamente implementadas;
- Executar todas as etapas do fluxo de projeto, até a geração do sistema final.

A.2 Repositório de Componentes

A suite de ferramentas permite que um novo repositório de componentes seja criado vazio, ou seja gerado a partir de um modelo de domínio, possivelmente especificado em UML numa ferramenta de engenharia de software. Nesse caso, diferentes implementações possíveis podem ser selecionadas para realizar a geração do repositório, como mostra a figura A.2.

Uma vez criado, o repositório possui a seguinte estrutura básica:

- Componentes. Um ou mais domínios, divididos em famílias de componentes, que agrupam um mais componentes membros;
- Software Tools. Um conjunto de ferramentas de software usadas para transformar os componentes, como compiladores e ferramentas de síntese, por exemplo, incluindo seus parâmetros e informações sobre tipos das entidades de entrada e de saída;
- Hardware packages: Um conjunto de encapsulamentos para componentes físicos, incluindo descrição de dimensões físicas, pesos e também pinagens, quando forem encapsulamentos de componentes eletro-eletrônicos;
- Hardware platforms. Um conjunto de “templates” de agrupamentos de componentes que especificam sub-circuitos eletrônicos ou suportes completos de hardware;
- System Quality Metrics: Um conjunto de características dos custos do sistema que podem ser usados para exploração do espaço de projeto.

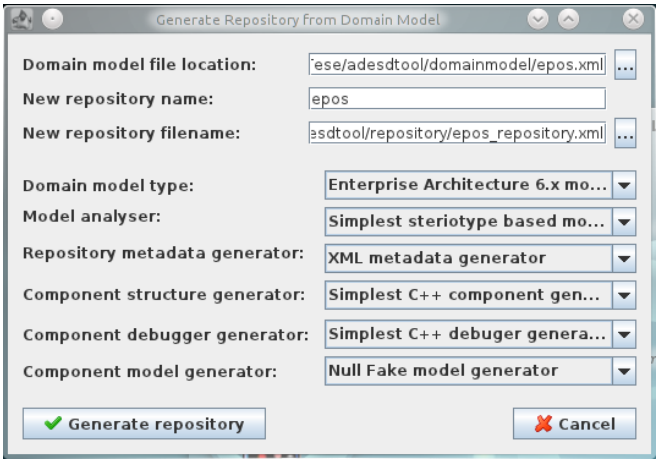


Figura A.2: *Geração do repositório a partir do modelo de domínio. O projetista pode escolher entre diferentes implementações das etapas necessárias para ler um modelo de domínio (descrito em UML, por exemplo) e gerar a estrutura de um repositório de componentes.*

A interface para visualização do repositório de componentes é apresentada nas figuras A.3 e figura A.4, que mostram um repositório vazio, recém criado, e um repositório já contendo componentes cadastrados, respectivamente.

O gerenciamento do repositório deve ser simples ao projetista de sistemas embarcados. A barra de ferramentas permite que sejam gerenciados (criado, editado, excluído) quaisquer elementos do repositório. Esses elementos aparecem hierarquicamente tanto na árvore de elementos á esquerda, quanto na área gráfica à direira, como ilustrado na figura A.4. As cores representam o tipo de componente (software, hardware sintetizável, arquitetura, etc) e também sua posição na área gráfica, que conecta também os componentes com base em suas dependências.

Cada tipo de componente e outros elementos do repositório (ferramentas, encapsulamentos, plataformas e caracterizações de custo) possui sua própria interface gráfica para cadastro e edição, das quais apenas poucas são ilustradas aqui. A figura A.5 apresenta algumas telas do cadastro de compo-

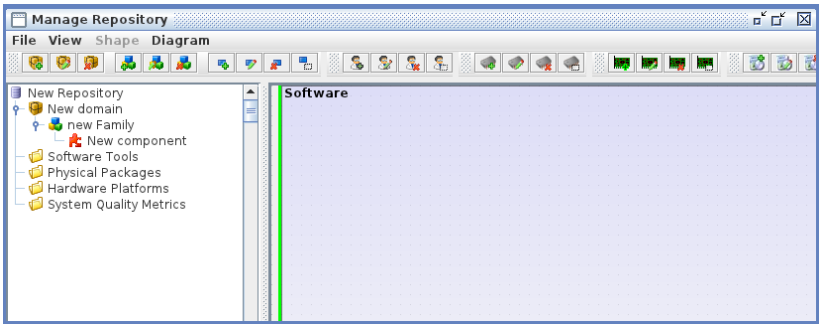


Figura A.3: *Repositório de componentes vazio. Interface gráfica de gerenciamento do repositório de componentes, apresentando um repositório vazio, recém criado.*

nentes. Nela, a figura A.5(a) apresenta as informações básicas de qualquer componente, como nome, família, tipo, *traits*, características configuráveis, dependências, documentação, etc. O botão "*Public Interface*" permite acessar a interface de cada tipo diferente de componente. A figura A.5(b) mostra, então, a interface pública de um componente de software, em que se especificam seus construtores, métodos, constantes, enumerações, definições de tipo e em quais arquiteturas esse componente pode ser mapeado. Como todo componente de software é também um componente *cyber*/lógico, o botão "*Define Logical Component Information*" permite definir informações que são comuns a qualquer componente lógico (software ou hardware sintetizável). A figura A.5(c) mostra essa tela. Nela, são especificadas a linguagem em que o componente foi implementado, complexidade e grau de maturidade, além do seu código-fonte e outros arquivos associados.

A figura A.6 apresenta telas análogas, para para o cadastro e edição de um componente de hardware físico que é, também, uma arquitetura. Nela, a figura A.6(a) mostra as informações básicas para definição da arquitetura, incluindo suas unidades de processamento, memória e periféricos, sendo que a figura A.6(c) mostra as informações específicas da unidade de processamento. A figura figura A.6(b) mostra as informações da interface de hardware, que incluem a especificação de pinos, barramentos e encapsulamentos físicos para esse componente, além de suas características elétricas. Por fim, a figura **??** mostra o cadastro de um pino dessa interface.

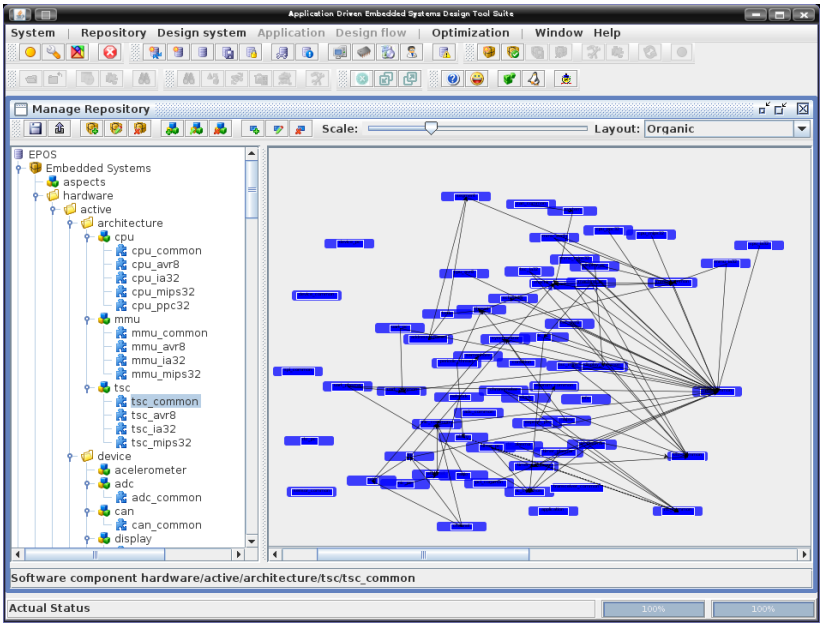
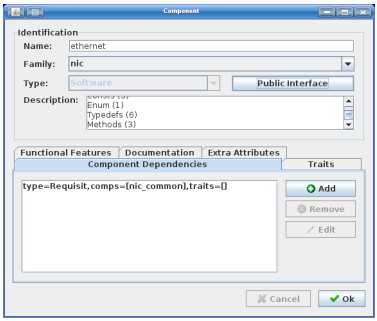
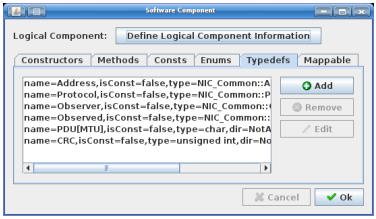


Figura A.4: *Gerenciamento do repositório de componentes. Interface gráfica de gerenciamento do repositório de componentes, apresentando um repositório com vários componentes já cadastrados.*

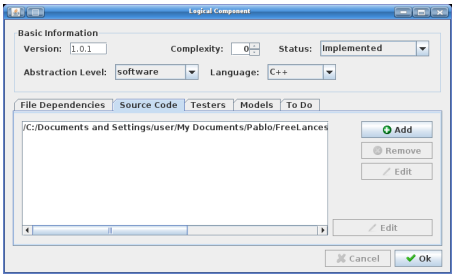
Como último exemplo do repositório, a figura A.7 mostra a tela de cadastro e edição de suportes de hardware. Todos os componentes de hardware, físico e sintetizável, aparecem na árvore à esquerda. Os componentes selecionados são, então, copiados à área gráfica à direita. Com o mouse o projetista pode fazer a interconexão elétrica entre esses componentes, definindo o esquemático de seu circuito eletrônico. Apesar da facilidade apresentada, certamente, muitos aperfeiçoamentos às interfaces gráficas ainda são necessários.



(a) Informações gerais de um componente



(b) Informações de um componente de software

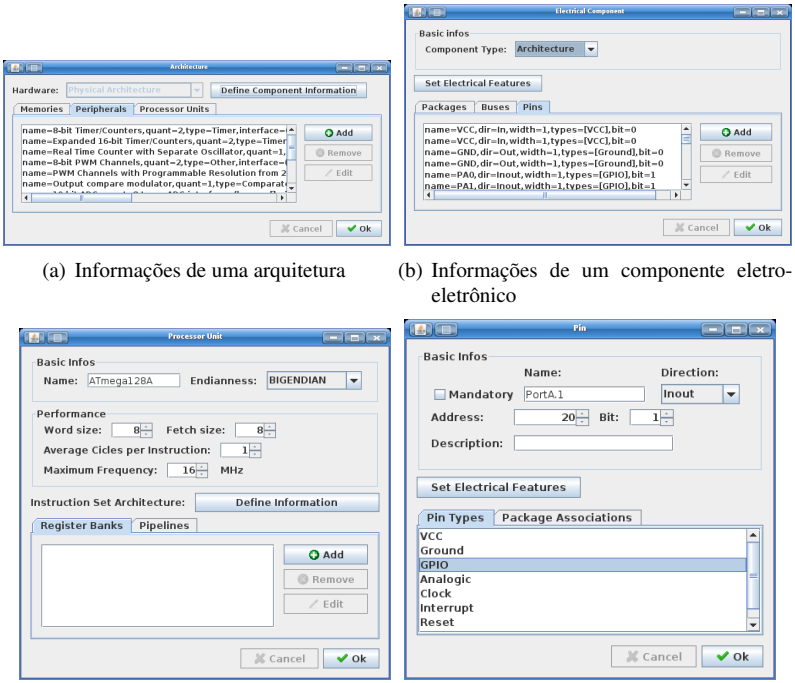


(c) Informações um componente lógico

Figura A.5: *Interfaces para cadastro de componentes cyber/lógicos*

A.3 Projeto de Sistemas Embarcados

O projeto de um sistema embarcado inicia com a definição de informações básicas sobre o projeto, conforme apresentado na figura A.8. Existem também cinco abas com informações de entrada para o projeto, conforme descrito no metamodelo de projeto de sistema embarcado, na seção 5.1. Pelo menos uma aplicação-alvo deve ser informada. O projetista pode informar ainda um conjunto de restrições de projeto, na forma de inequações sobre caracterizações do sistema, um conjunto de aspectos a serem aplicados sobre os componentes da solução, um conjunto de regiões de interesse, na forma de equações ou distribuições de probabilidade sobre caracterizações do sis-



(a) Informações de uma arquitetura

(b) Informações de um componente eletro-eletrônico

(c) Informações uma unidade de processamento

(d) Informações um pino de conexão

Figura A.6: *Interfaces para cadastro de componentes físicos*

tema, e um conjunto de projetos relacionados de alguma forma, para que sejam aproveitadas suas soluções para compor parte da população inicial na exploração do espaço de projeto atual.

A.4 Fluxo de Projeto e Exploração do Espaço de Projeto

Após a especificação do projeto do sistema embarcado, o projetista precisa apenas proceder com a execução das etapas do fluxo de projeto. Ainda não há interface gráfica para a visualização de cada etapa desse fluxo, e al-

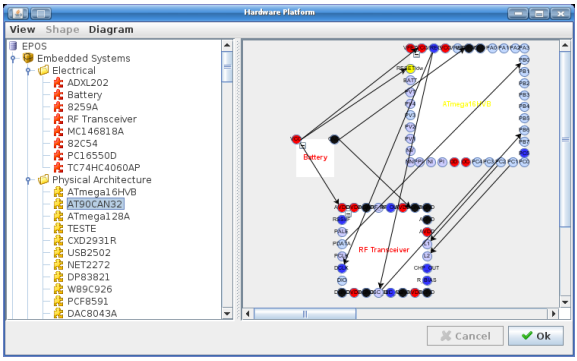


Figura A.7: *Interface para cadastro de plataformas de hardware*

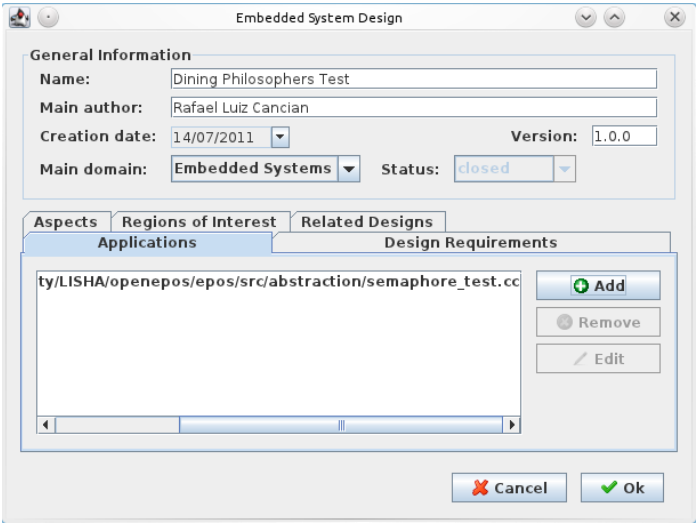
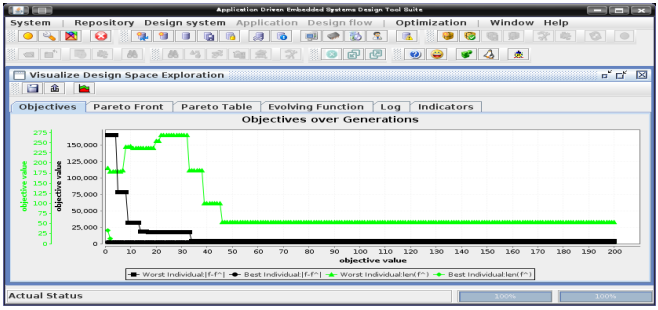


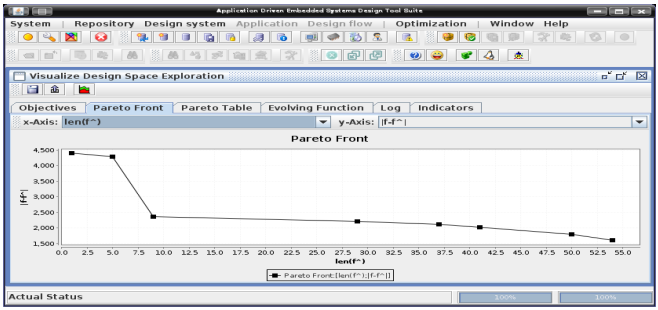
Figura A.8: *Informações do Projeto de um Sistema Embarcado. Ao projetar um novo sistema, o projetista fornece informações básicas e também cinco outras entradas: as aplicações-alvo, restrições de projeto, regiões de interesse do projetista, aspectos a serem aplicados e projetos relacionados.*

gumas etapas realmente não geram informação que possa ou deva ser apre-

sentada de forma gráfica. Durante essa etapa, em geral, apenas *logs* gerados pelas ferramentas são apresentados na tela de logs (ver figura A.1), em forma textual.



(a) Evolução dos objetivos de exploração ao longo das iterações



(b) Fronteira de Pareto com as soluções encontradas na exploração do espaço de projeto

Figura A.9: *Gráfico da fronteira de Pareto. Interface gráfica dos resultados da otimização multiobjetivo com a fronteira de Pareto*

Na versão atual da interface gráfica da suite de ferramentas da ADESD, os resultados finais do projeto de um sistema embarcado são mostrados numa tela simples, que apresenta um resumo dos experimentos realizados, a evolução dos objetivos ao longo das iterações, a fronteira de Pareto com as melhores soluções encontradas, uma tabela com a relação dessas soluções e indicadores de qualidade. Para ilustrar isso, A figura A.9(b) apresenta a tela com a evolução dos objetivos e a figura A.9(a) apresenta a tela com a

fronteira de Pareto.

Embora o desenvolvimento da suite de ferramentas da ADESD, em si, não tenha foco desta tese, ela está em estágio avançado de desenvolvimento e deve ser alvo de desenvolvimento pelos integrantes do Laboratório de Integração de Software e Hardware nos próximos meses.

APÊNDICE B

REPOSITÓRIO DE COMPONENTES PARA SISTEMAS EMBARCADOS

O repositório de componentes da suite de ferramentas da ADESD foi desenvolvido a partir do metamodelo de componentes apresentado nesta tese. Além de (1) Componentes físicos (eletro-eletrônicos, mecânicos) e lógicos (software, hardware sintetizável), o repositório descreve informações sobre (2) Plataformas de hardware (físico ou sintetizável), que podem ser usadas para suportar aplicações embarcadas, ou servirem de base à evolução e criação de novas plataformas; sobre (3) Encapsulamentos físicos, que descrevem a interface de componentes físicos (principalmente componentes eletro-eletrônicos); (4) Sobre ferramentas de software, que são usadas nos processos de geração e verificação de software e hardware sintetizável (compiladores, ligadores, sintetizadores, depuradores, simuladores, etc); e também sobre (5) Métricas de qualidade do sistema, que representam possíveis funções-objetivo a serem otimizadas no processo de exploração do espaço de projeto.

Este anexo apresenta o repositório atual da suite de ferramentas da ADESD, populado com alguns componentes pré-existentes (componentes de software) e também com novos componentes criados durante esta tese, como os componentes físicos e de hardware sintetizável, ferramentas de software, encapsulamentos físicos e também as plataformas de hardware. Inicialmente, a seção B.1 é apresentada a estrutura geral do repositório, depois a seção B.2 apresenta uma lista simplificada dos componentes do repositório, separados por tipo, e então a seção B.3 uma lista completa, mostrando as características principais de cada componente e que são separados pela estrutura de família (conforme a FbD) a que pertencem.

B.1 Estrutura Geral do Repositório

B.2 Relação dos Componentes por Tipo

- Software

  - channel, periodic_thread, uart_pc, ethernet, eeprom_common, cpu_avr8, eeprom_pc, timer_plasma, uart_plasma, device_plasma, nic_common, mmu_avr8, tsc_avr8, tsc_ia32, task, tsc_common, network, ic_pc, cpu_mips32, transceiver_common, ip, application, sensor_common, synchronizer, address_space, clock, machine_pc, machine_common, ic_plasma, timer_common, can_common, rtc_plasma, cpu_common, display_common, segment, rtc_pc, adc_common, radio, ic_common, uart_common, cpu_ppc32, rtc_common, spi_common, udp, semaphore, system, alarm, pci_common, nic_pc, device_common, display_pc, condition, tsc_mips32, machine_plasma, device_pc, info, timer_pc, cpu_ia32, thread, mmu_mips32, mutex, mmu_common, chronometer, scheduler, mmu_ia32

- Mechanical
  -

- Electrical
  - MC146818A, RF Transceiver, 8259A, 82C54, ADXL202, PC16550D, TC74HC4060AP

- Synthesizable
  - OCPWrapper, AIX4LITE, Timer_IP, IC_IP, USART_IP

- Physical Architecture
  - AT90CAN32, TESTE, ATmega16HVB, ATmega128A

- Programmable Logic Device
  - XC3S1200E-4FG400C, XC3S100E-4TQ144C, XC6VLX760-2FFG1760C, XC6VLX75T-1FF484C, XC3S250E-4FT256C, XC6SLX75-N3FGG676I, XC6VLX365T-2FFG1759C, XC3S500E-4FT256C, XC6SLX45-N3FGG676C, XC4VLX80-10FF1148C, XC6VLX75T-2FFG784I, XC3195, XC6SLX100-N3FGG676I, XC6SLX16-2CSG324CES, XC5202, XC3020, XC4VLX60-10FF1148C, XC5215, XC4VLX25-11FF668C, XC6SLX150-N3FGG900I, XC4VLX100-10FF1148C, XC6VLX240T-2FFG784CES, XC6VLX195T-1FF1156CES

- Synthesizable Architecture
  - PLASMA
- Physical Packages
  - GenericPackage, FF1148, FF668, FF484, FF784, FF1759, FF1760, FFG1156, FGG676, FGG900, FG400, FT256, FBGA 1152 pin A:3.30, FBGA 1152 pin A:3.40, FBGA 1517 pin A:3.30, FBGA 1517 A:3.40, FBGA 672 pin, FBGA 869 pin, FP-64A, FP-64E, FP-48F, FP-80A, FP-100A, FP-100B, FP-100U, HCB0/HIB0 TBGA 63 pin, HBGA 780 pin A:3.40, MLF (VDFN) 10, PQFP 208, MLF (WQFN) 20, MLF (VQFN) 28, MLF (VQFN) 32, MLF (VQFN) 64, MLF (VQFN) 44, LQFP-144, NanoBGA2, TSOP 48, QFN 36, PDIP 14, PDIP 20, PDIP 8, SOIC (300mil) 20, SOIC (150mil) 8, SOIC (208mil) 8, PDIP 40, PDIP 28, SOIC (150mil) 14, QFN 44, QFN 64, SOIC 20, SOIC 8, 88_Lead_LFCSP, 176-Lead_LQFP_Ep , 168-Ball_CSP_BGA, 160-Ball_CSP_BGA, 169-Ball_PBGA, 176-Lead_LQFP, 400-Ball_CSP_BGA, 289-Ball_CSP_BGA, 208-Ball_CSP_BGA, 368-Ball_EBGA, 100-Lead_PQFP, 100-CBGA, 368-EBGA, 324-Ball_BGA, 63-Ball_TBGA, 217-Ball_LFBGA, 144-Ball_LFBGA, FCB0_WSOP, DP-42S, DP-64S, DIP-16, 144-Pin_EQFP, 484-Pin_FBGA, 324-Pin_FBGA, 256-Pin_FBGA, DIP8, SOIC8, SOT23-6, SO-16, TFBGA144, TFBGA324, TSSOP14, TSSOP20, TQFP44, TQFP64, TQ144, TQFP48, TQFP100, TQFP32, UFBGA15, UFBGA32, UBGA484, VFBGA49, VQFN20, VQFN (Sawn) 32, VQFN (Sawn) 64, TSOP48, WSOP48, TFP-80C, TFP-100B, DFN8, UDFN/USON-8
- System Quality Metrics
  - physical_area, memory, pins, weight, frequency
- Software Tools
  - gcc402, gcc444, g++402, g++444, avrgcc402, avrg++404, mipsgcc402, mipsg++402
- Hardware platforms
  - eposmote2, arduino_bt6, arduino_uno

B.3 Relação dos Componentes e Características

- aspects
- hardware/active/architecture/cpu
  - cpu_common (Software). Methods=[halt, tsl, finc, fdec, cas]. Typedefs=[Reg16, Reg32, Reg64, Reg8, Phy_Addr, Hertz]
  - cpu_avr8 (Software). Dependences=[cpu_common]. Methods=[save, load, int_enable, int_disable, halt, switch_context, flags, flags, sp, sp, fr, fr, pdp, pdp, ip, tsl, finc, fdec, htonl, htons, ntohl, ntohs, init_stack, init_stack, init_stack, int_stack, Clock, sreg, sphl, sreg, sphl, r25_24, r25_24, pc, in8, in16, out8, out16, power, power, reboot, init, clock]. Enums=[,]. Typedefs=[Flags]
  - cpu_ia32 (Software). Dependences=[cpu_common]. Methods=[IDT_ENTRIES, clock, bus_clock, int_enable, int_disable, halt, switch_context, flags, flags, sp, sp, fr, fr, pdp, pdp, ip, tsl, finc, fdec, cas, htonl, htons, ntohl, ntohs, init_stack, init_stack, init_stack, init_stack, init, eflags, eflags, esp, esp, eax, eax, eip, cr0, cr0, cr2, cr3, cr3, gdtr, gdtr, idtr, idtr, cs, ds, es, ss, fs, gs, in8, in16, in32, out8, out16, out32, switch_tss]. Enums=[, Exceptions, , , GDT_Layout, ,]. Consts=[IDT_ENTRIES]. Typedefs=[Flags, IO_Port, IO_Irq, ISR, FSR]
  - cpu_mips32 (Software). Dependences=[cpu_common]. Methods=[clock, int_enable, switch_context, int_disable, halt, tsl, finc, fdec, htonl, htons, ntohl, ntohs, cpu_to_le32, cpu_to_le16, le32_to_cpu, le16_to_cpu, init_stack, init_stack, init_stack, init_stack, entry_wrapper, entry_wrapper, entry_wrapper, entry_wrapper, sp, sp, fr, fr, pdp, init]. Enums=[STATUS_REG, CAUSE, CP]. Consts=[STATUS_INT_MASK]. Typedefs=[CPOReg]
  - cpu_ppc32 (Software). Dependences=[cpu_common]. Methods=[clock, switch_context, int_enable, int_disable, halt, tsl, finc, fdec, htonl, htons, ntohl, ntohs, cpu_to_le32, cpu_to_le16, le32_to_cpu, PPC32::init_stack, PPC32::init_stack, PPC32::init_stack, PPC32::init_stack, entry_wrapper, entry_wrapper, entry_wrapper, entry_wrapper, fr, fr, sp, sp, pdp, pdp, init, _mfspr, _mtspr, sync_io]. Enums=[MSR, ESR, SPR].

Typedefs=[IO_Port, IO_Irq]
  - ATmega16HVB (Physical Architecture)
  - AT90CAN32 (Physical Architecture)
  - ATmega128A (Physical Architecture)
  - TESTE (Physical Architecture). Dependences=[mmu]. Attributes=[ExtraAttribute1]. Traits=[Trait1]. features=[Feature1]

- hardware/active/architecture/mmu
  - mmu_common (Software). Dependences=[cpu_common]. Methods=[pages, page_tables, offset, indexes, page, directory, aling32, aling64, aling128, aling_page, aling_directory]. Consts=[PAGE_SHIFT, DIRECTORY_SHIFT, PAGE_SIZE, PT_ENTRIES, PD_ENTRIES]. Typedefs=[Log_Addr, Phy_Addr, Page[PAGE_SIZE], Frame, PT_Entry, PD_Entry]
  - mmu_avr8 (Software). Dependences=[mmu_common, cpu_common]. Methods=[flush_tlb, flush_tlb, alloc, calloc, free, current, physical]. Typedefs=[AVR8_Flags, Page_Directory]
  - mmu_ia32 (Software). Dependences=[cpu_common, mmu_common]. Methods=[IA32_Flags, IA32_Flags, IA32_Flags, alloc, calloc, free, current, physical, flush_tlb, flush_tlb, init]. Typedefs=[Page_Directory]
  - mmu_mips32 (Software). Dependences=[cpu_common, mmu_common]. Methods=[flush_tlb, flush_tlb, alloc, calloc, free, current, physical, init]. Typedefs=[Page_Directory]

- hardware/active/architecture/tsc
  - tsc_common (Software). Typedefs=[Hertz, Time_Stamp]
  - tsc_avr8 (Software). Dependences=[cpu_common, tsc_common]. Methods=[frequency, frequency, time_stamp, init, _ts]
  - tsc_ia32 (Software). Dependences=[cpu_common, tsc_common]. Methods=[frequency, time_stamp]
  - tsc_mips32 (Software). Dependences=[cpu_common, tsc_common]. Methods=[frequency, time_stamp]

- hardware/active/device/acelerometer
  - ADXL202 (Electrical)

- hardware/active/device/adc
  - adc_common (Software). Dependences=[]. Enums=[Channel, Reference, Trigger]
- hardware/active/device/can
  - can_common (Software). Dependences=[system]. Enums=[]
- hardware/active/device/display
  - display_common (Software). Dependences=[uart_common]
  - display_pc (Software). Dependences=[display_common]. Methods=[remap, putc, puts, clear, position, position, geometry]. Enums=[]. Typedefs=[Cell, Attribute]
- hardware/active/fpga
  - XC5202 (Programmable Logic Device)
  - XC5215 (Programmable Logic Device)
  - XC3020 (Programmable Logic Device)
  - XC3195 (Programmable Logic Device)
  - XC6VLX75T-1FF484C (Programmable Logic Device)
  - XC6VLX75T-2FFG784I (Programmable Logic Device)
  - XC6VLX365T-2FFG1759C (Programmable Logic Device)
  - XC6VLX240T-2FFG784CES (Programmable Logic Device)
  - XC6VLX195T-1FF1156CES (Programmable Logic Device)
  - XC6VLX760-2FFG1760C (Programmable Logic Device)
  - XC4VLX25-11FF668C (Programmable Logic Device)
  - XC4VLX60-10FF1148C (Programmable Logic Device)
  - XC4VLX80-10FF1148C (Programmable Logic Device)
  - XC4VLX100-10FF1148C (Programmable Logic Device)
  - XC6SLX16-2CSG324CES (Programmable Logic Device)
  - XC6SLX100-N3FGG676I (Programmable Logic Device)
  - XC6SLX150-N3FGG900I (Programmable Logic Device)
  - XC6SLX45-N3FGG676C (Programmable Logic Device)
  - XC6SLX75-N3FGG676I (Programmable Logic Device)
  - XC3S100E-4TQ144C (Programmable Logic Device)
  - XC3S1200E-4FG400C (Programmable Logic Device)
  - XC3S250E-4FT256C (Programmable Logic Device)
  - XC3S500E-4FT256C (Programmable Logic Device)

- hardware/active/interconection/noc
- hardware/active/interconection/ocb
- hardware/active/interconection/wrapper
- hardware/machine
  - machine_common (Software)
  - machine_plasma (Software). Dependences=[cpu_common, mmu_common, tsc_common, machine_common]. Methods=[int_vector, int_vector, seize, release, irq2int, int2irq, panic, reboot, poweroff, init]. Typedefs=[int_handler]
  - machine_pc (Software). Dependences=[]. Methods=[panic, reboot, poweroff, n_cpus, cpu_id, smp_init, smp_barrier, init]
- hardware/passive/battery
- hardware/passive/connector
- system/abstract/communication/channel
  - channel (Software). Dependences=[ip, nic_common, udp]. Methods=[send, receive]. Typedefs=[Address]
- system/abstract/communication/communicator
- system/abstract/synchronizer
  - condition (Software). Dependences=[synchronizer]. Methods=[wait, signal, broadcast]
  - mutex (Software). Dependences=[synchronizer]. Methods=[lock, unlock]
  - synchronizer (Software). Dependences=[cpu_common, thread]
  - semaphore (Software). Dependences=[synchronizer]. Methods=[p, v]
- system/abstract/time
  - alarm (Software). Dependences=[rtc_common]. Methods=[resolution, delay, init]. Enums=[]. Typedefs=[Hertz , Microsecond]

  - chronometer (Software). Dependences=[tsc_common, rtc_common]. Methods=[frequency, reset, start, lap, , ticks, read]. Typedefs=[Hertz, Time_Stamp, Microsecond]
  - clock (Software). Dependences=[rtc_common]. Methods=[resolution, now, date, date]. Typedefs=[Microsecond, Second, Date]

- system/abstract/process/scheduler
  - scheduler (Software). Dependences=[cpu_common, machine_common]. Methods=[schedulables, chosen, insert, remove, suspend, resume, choose, choose_another, choose]. Typedefs=[Criterion, Queue, Element]

- system/abstract/process/task
  - task (Software). Dependences=[address_space, thread, segment]. Methods=[as, code, data, create_thread, destroy_thread, init]

- system/abstract/process/thread
  - thread (Software). Dependences=[cpu_common, scheduler]. Methods=[state, criterion, priority, priority, join, pass, suspend, resume, self, yield, sleep, wakeup, wakeup_all, exit, init]. Enums=[State,]. Typedefs=[Priority, Criterion, Queue]
  - periodic_thread (Software). Dependences=[thread, alarm]. Methods=[wait_next]

- system/abstract/communication/device

- system/abstract/communication/envelope
  - ip (Software). Dependences=[cpu_common, nic_common, ethernet]. Methods=[received, self, attach, get_fragment]. Enums=[]. Consts=[MTU, BROADCAST, logic_address]. Typedefs=[u8, u16, u32, MAC_Address, Protocol, Statistics, ARP]
  - udp (Software). Dependences=[ip, semaphore]. Methods=[received, receive, self]. Consts=[ID_UDP]

- system/abstract/communication/network
  - network (Software). Dependences=[machine_common, nic_common, ethernet]. Methods=[send, receive, arp,

rarp, update, reset, address, statistics]. Enums=[]. Consts=[BROADCAST]. Typedefs=[MAC_Address, Protocol, Statistics, ARP]

- system/abstract/io/bus
- system/abstract/memory/address_space
  - address_space (Software). Dependences=[mmu_common, segment]. Methods=[attach, attach, detach, active, physical]. Enums=[Self]
- system/abstract/io/interrupt_handler
- system/abstract/memory/segment
  - segment (Software). Dependences=[mmu_common]. Methods=[size, phy_address, resize]. Typedefs=[Flags, Phy_Addr]
- system/boot
- system/info
  - system (Software). Dependences=[machine_common]. Methods=[info, heap, init]
  - application (Software). Methods=[heap, init]
  - info (Software)
- system/init
- system/setup
- hardware/active/device/dma
- hardware/active/device/eeprom
  - eeprom_common (Software). Typedefs=[Address]
  - eeprom_pc (Software). Dependences=[eeprom_common, rtc_common]. Methods=[read, write, size]. Typedefs=[Address]
- hardware/active/device/gps
- hardware/active/device/ic

  - ic_common (Software). Typedefs=[Interrupt_Id, Interrupt_Handler]
  - ic_plasma (Software). Dependences=[cpu_common, ic_common]. Methods=[enable, mask, disable, disable, int_handler, init]. Enums=[, IC_STATUS_REG]. Consts=[DEFAULT_ENABLE_INTR_MASK]
  - ic_pc (Software). Dependences=[cpu_common, ic_common]. Methods=[int_vector, int_vector, enable, enable, disable, disable, init]
  - 8259A (Electrical)

- hardware/active/device/nic
  - ethernet (Software). Dependences=[nic_common]. Methods=[attach, detach, notify]. Enums=[]. Consts=[MTU, HEADER_SIZE, BROADCAST]. Typedefs=[Address, Protocol, Observer, Observed, PDU[MTU], CRC]
  - nic_common (Software). Dependences=[cpu_common]. Typedefs=[Protocol, Observer, Observed]
  - radio (Software). Dependences=[nic_common]. Methods=[attach, detach, notify]. Enums=[]. Consts=[MTU, HEADER_SIZE, TRAILER_SIZE, BROADCAST]. Typedefs=[Address, Protocol]
  - nic_pc (Software). Dependences=[ethernet]. Methods=[send, recieve, reset, mtu, address, statistics, init]

- hardware/active/device/pci
  - pci_common (Software). Dependences=[cpu_common]. Enums=[, , , , , PCI_Masks, , , , , ,]. Typedefs=[Class_Id, Vendor_Id, Device_Id]

- hardware/active/device/radio
  - transceiver_common (Software). Enums=[result_t, event_t]. Typedefs=[microseconds_t, (event_handler)(event_t)]
  - RF Transceiver (Electrical)

- hardware/active/device/rtc
  - rtc_common (Software). Typedefs=[Microsecond, Second]

  - rtc_plasma (Software). Dependences=[rtc_common]. Methods=[date, date, seconds_since_epoch]
  - rtc_pc (Software). Dependences=[]. Methods=[reg, reg, cmos_read, cmos_write, cmos_size]. Enums=[, , , , ,]. Typedefs=[Address]
  - MC146818A (Electrical)

- hardware/active/device/sensor
  - sensor_common (Software)

- hardware/active/device/spi
  - spi_common (Software)

- hardware/active/device/timer
  - timer_common (Software). Dependences=[tsc_common]. Typedefs=[Hertz, Tick, Handler]
  - timer_plasma (Software). Dependences=[cpu_common, ic_common, timer_common]. Methods=[frequency, frequency, enable, disable, reset, int_handler, init]
  - timer_pc (Software). Dependences=[cpu_common, ic_common, rtc_common, timer_common]. Methods=[frequency, frequency, read, reset, handler, enable, disable, init]. Enums=[]. Typedefs=[Channel]
  - 82C54 (Electrical)

- hardware/active/device/uart
  - uart_common (Software)
  - uart_plasma (Software). Dependences=[uart_common, cpu_common, ic_common]. Methods=[get, put, reset, loopback, int_enable, int_disable, power, power, int_handler, init]. Enums=[IRQ_UART,]
  - uart_pc (Software). Dependences=[cpu_common, uart_common]. Methods=[config, config, get, put, loopback, power, power]. Enums=[]
  - PC16550D (Electrical)

- hardware/passive/misc
  - TC74HC4060AP (Electrical)

- hardware/passive/pcb
- hardware/passive/rlc
- hardware/active/device/device
  - device_plasma (Software). Methods=[object, seize, release, get, get, install_handler]
  - device_common (Software)
  - device_pc (Software). Methods=[object, seize, release, get, get, install_handler]
- system/memory_map

GLOSSÁRIO

alocação	indica quais elementos da arquitetura são usados na implementação	p. 39
escalonamento	associa um tempo de início de execução a cada elemento de software	p. 39
arquitetura	a interface do hardware que é visível em nível de software e aos desenvolvedores do sistema, e que constitui-se, basicamente das instruções e registradores reconhecidos pela CPU e pelos mecanismos de acesso aos dispositivos	p. 4
caracterização de custo do sistema	característica do sistema sendo projetado e que pode ser utilizada como função-objetivo a ser otimizada	p. 87
co-simulação	simulação integrada de software e hardware	p. 2
codificação	corresponde ao mapeamento entre as variáveis do problema e a representação computacional de um vetor de símbolos, que representam os genes de um indivíduo	p. 26
codon	seqüência composta por 3 nucleotídeos e que forma o código que corresponde a um único aminoácido na tradução de uma proteína	p. 21

componente híbrido	componente que podem ser mapeado tanto para software quanto para hardware	p. 42
conjunto de Pareto	conjunto de soluções que não são *dominadas* por nenhuma outra solução do *espaço de soluções*	p. 11
critério de parada	define as condições no qual assume-se que uma solução ótima global foi encontrada e o processamento pode ser encerrado	p. 24
cromossomo	é um vetor de genes relacionados de alguma forma, e normalmente associados à mesma função-objetivo	p. 23
deceptiva	função-objetivo multimodal no qual a maior parte do espaço de busca leva ao mínimo local deceptivo	p. 17
deriva genética	processo estocástico, atuante sobre as populações, que modifica a frequência dos alelos e a predominância de certas características na população	p. 24
diversidade da população	é uma medida da variação entre os indivíduos da mesma população, e é representada pelo desvio-padrão da função de aptidão da população	p. 24
elitismo	garantia da inclusão das melhores soluções da população atual da próxima população, preservando as melhores soluções já encontradas	p. 28

epigenética	refere-se a mudanças moleculares que regulam a função de genes, mas que não envolvem mudanças no material genético	p. 22
epistasia	interação entre genes, causada quando um gene é modificado por um ou muitos outros genes, chamados genes modificadores	p. 26
evolução	mudança da frequência relativa de um gene na população	p. 3
fenótipo	é o efeito das variáveis do problema em relação aos aspectos de interesse, representados pelos valores das funções-objetivo	p. 24
função de aptidão	mapeia cada indivíduo da população atual a uma oportunidade de reprodução	p. 23
genótipo	é o vetor de símbolos que representam seus genes	p. 24
indivíduo	é um vetor de cromossos, de forma que um indivíduo representa, de forma codificada, uma possível solução ao problema	p. 23
intensidade de seleção	diferença da aptidão média da população ($\bar{\phi}$) após a seleção e antes da seleção	p. 24
metaprogramação	paradigma de programação no qual os algoritmos são escritos em uma gramática estendida de forma a adaptar-se através da especificação das partes variáveis que são definidas na instância do algoritmo	p. 36

multimodal	função-objetivo que apresenta múltiplos mínimos locais	p. 17
nichos	sub-populações razoavelmente estáveis, análogas a povoados geograficamente isolados	p. 28
operador de recombinação	define como o genótipo de indivíduos-pais é recombinado para formar o genótipo de novos indivíduos-filhos	p. 29
operadores de seleção	são usados para escolher, dentre os indivíduos da população atual, aqueles que serão usados para gerar a próxima geração da população	p. 27
operadores de variação	definem o modo no quais indivíduos-pais geram indivíduos-filhos (*offsprings*) e como alterações genômicas aleatórias são aplicadas	p. 28
operadores ordinais	operadores de seleção que escolhem indivíduos com base em sua ordenação relativa na população, e não diretamente em sua aptidão	p. 27
operadores proporcionais	operadores de seleção em que a probabilidade de um indivíduo ser selecionado para reprodução é proporcional à sua aptidão	p. 27
plataforma	uma abstração que cobre vários possíveis refinamentos em níveis mais baixos	p. 33
ponto de nadir	limite superior de cada função-objetivo no conjunto completo de Pareto	p. 18

ponto ideal	valor mínimo de cada função-objetivo individualmente num espaço de M dimensões	p. 18
população	é um conjunto de indivíduos de certa geração	p. 23
pressão de seleção	grau no qual apenas os melhores indivíduos são escolhidos, e está diretamente relacionada à diferença entre a aptidão dos melhores e dos piores indivíduos	p. 24
rede metabólica	série de reações químicas onde uma reação fornece o substrato da reação seguinte sendo a reação seguinte dependente da anterior	p. 22
solução viável	uma solução que não viola nenhuma das restrições impostas	p. 9
suporte de hardware	ligação de componentes de hardware, de modo que formem um circuito eletrônico contendo pelo menos uma unidade de processamento e que forma a base sobre a qual executa o software	p. 5
sítio ativo	uma região de uma proteína que participa da interação dessa proteína com outros compostos, e é responsável pela manutenção da funcionalidade da proteína no organismo	p. 22
template de circuito	agregação de componentes de hardware físico ou sintetizável, fixos e interconectados de maneira específica, e que podem ser reusados em diferentes circuitos e suportes de hardware	p. 57

unimodal	função-objetivo que apresenta um único mínimo local	p. 17
variação da população	diferença entre a aptidão média de uma população e de populações passadas	p. 24
ótimo isolada	função-objetivo no qual boa parte de seu conjunto de Pareto é plano, não fornecendo informação útil sobre o ponto ótimo	p. 17

REFERÊNCIAS BIBLIOGRÁFICAS

[Ahn et al. 2010]AHN, Y. et al. Novel Memetic Algorithm implemented With GA (Genetic Algorithm) and MADS (Mesh Adaptive Direct Search) for Optimal Design of Electromagnetic System. *Magnetics, IEEE Transactions on*, IEEE, v. 46, n. 6, p. 1982–1985, jun. 2010. ISSN 0018-9464. <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5467565>.

[Al Rayahi e Khalid 2009]Al Rayahi, O. a.; KHALID, M. a. S. UWindsor Nios II: A soft-core processor for design space exploration. *2009 IEEE International Conference on Electro/Information Technology*, Ieee, p. 451–457, jun. 2009. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5189659>.

[Anderson e Khalid 2006]ANDERSON, I. D. L.; KHALID, M. A. S. Design space exploration using parameterized cores: A case study. In: *CCECE '06: Proceedings of the Canadian Conference on Electrical and Computer Engineering*. [S.l.: s.n.], 2006. p. 1893–1896.

[Ascia et al. 2006]ASCIA, G. et al. An efficient hierarquical fuzzy approach for system-level design space exploration. In: *IEEE International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. [S.l.: s.n.], 2006. p. 115–122.

[Ascia et al. 2007]ASCIA, G. et al. Efficient design space exploration for application specific systems-on-a-chip. *Journal of Systems Architecture*, p. 733–750, 2007.

[Ashlock 2006]ASHLOCK, D. *Evolutionary Computation for Modeling and Optimization*. [S.l.]: Springer Science & Business Media, 2006. ISBN 978-0387-22196-0.

[Auger et al. 2009]AUGER, A. et al. Articulating user preferences in many-objective problems by sampling the weighted hypervolume. In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. New York, New York, USA: ACM, 2009. p. 555–562. ISBN 9781605583259. <http://portal.acm.org/citation.cfm?id=1569979>.

[Auger et al. 2009]AUGER, A. et al. Theory of the hypervolume indicator. *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic*

algorithms - FOGA '09, ACM Press, New York, New York, USA, p. 87, 2009. <http://portal.acm.org/citation.cfm?doid=1527125.1527138>.

[Azizi et al. 2010]AZIZI, O. et al. An Integrated Framework for Joint Design Space Exploration of Microarchitecture and Circuits. *Pace Pacing And Clinical Electrophysiology*, 2010.

[Bader, Brockhoff e Zitzler 2011]BADER, J.; BROCKHOFF, D.; ZITZLER, E. *Weighted Hypervolume Indicator*. [S.l.], Mai 2011.

[Bader e Zitzler 2008]BADER, J.; ZITZLER, E. HypE : An Algorithm for Fast Hypervolume-Based. *Computer Engineering*, 2008.

[Bader 2009]BADER, J. M. *Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods*. Tese (Doutorado) — Swiss Federal Institute of Technology Zurich, December 2009.

[Balarin et al. 1997]BALARIN, F. et al. *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. [S.l.]: Springer, 1997. (The Springer International Series in Engineering and Computer Science, v. 404).

[Balarin et al. 2003]BALARIN, F. et al. Metropolis: an integrated electronic system design environment. *Computer*, v. 36, n. 4, p. 45–52, abr. 2003. ISSN 0018-9162. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1193228>.

[Bauer, Shafique e Henkel 2009]BAUER, L.; SHAFIQUE, M.; HENKEL, J. Cross-Architectural Design Space Exploration Tool for Reconfigurable Processors. In: *Design, Automation and Test in Europe*. Nice, France: [s.n.], 2009. p. 958–963.

[Beltrame et al. 2006]BELTRAME, G. et al. Decision-theoretic exploration of multiprocessor platforms. In: *CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2006. p. 205–210. ISBN 1-59593-370-0.

[Benedetti, Farina e Gobbi 2006]BENEDETTI, a.; FARINA, M.; GOBBI, M. Evolutionary multiobjective industrial design: the case of a racing car tire-suspension system. *IEEE Transactions on Evolutionary Computation*, v. 10, n. 3, p. 230–244, jun. 2006. ISSN

1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1637685>.

[Bergamaschi et al. 2008]BERGAMASCHI, R. et al. The State of ESL Design [Roundtable]. *IEEE Design & Test of Computers*, v. 25, n. 6, p. 510–519, nov. 2008. ISSN 0740-7475. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4702875>.

[Beume et al. 2009]BEUME, N. et al. On the Complexity of Computing the Hypervolume Indicator. *IEEE Transactions on Evolutionary Computation*, v. 13, n. 5, p. 1075–1082, out. 2009. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5208224>.

[Bjerregaard e Mahadevan 2006]BJERREGAARD, T.; MAHADEVAN, S. A survey of research and practices of network-on-chip. In: *ACM Computing Surveys*. [S.l.]: ACM New York, 2006. v. 38.

[Bleuler et al. 2003]BLEULER, S. et al. PISA - A Platform and Programming Language Independent Interface for Search Algorithms. *Interface*, p. 494–508, 2003.

[Bonissone e Subbu 2007]BONISSONE, S.; SUBBU, R. Evolutionary Multiobjective Optimization on a Chip. n. Weah, p. 61–66, 2007.

[Bradstreet, Barone e While 2009]BRADSTREET, L.; BARONE, L.; WHILE, L. Updating exclusive hypervolume contributions cheaply. *2009 IEEE Congress on Evolutionary Computation*, Ieee, p. 538–544, maio 2009. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4982992>.

[Bradstreet, While e Barone 2008]BRADSTREET, L.; WHILE, L.; BARONE, L. A Fast Incremental Hypervolume Algorithm. *IEEE Transactions on Evolutionary Computation*, v. 12, n. 6, p. 714–723, dez. 2008. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4481244>.

[Calborean e Vintan 2010]CALBOREAN, H.; VINTAN, L. An automatic design space exploration framework for multicore

architecture optimizations. In: *Roedunet International Conference (RoEduNet), 2010 9th*. IEEE, 2010. p. 202–207. <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5541567>.

[Cancian, Fröhlich e Stemmer 2007]CANCIAN, R. L.; FRöHLICH, A. A. M.; STEMMER, M. R. New developments in epos tools for configuring and generating embedded systems. In: *Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation*. Patras, Greece: [s.n.], 2007. p. 776–779.

[Cancian, Stemmer e Fröhlich 2006]CANCIAN, R. L.; STEMMER, M. R.; FRöHLICH, A. A. M. Real-time domain engineering for soc-design. In: *Proceedings of The 27th IEEE Real-Time Systems Symposium - WiP section*. Rio de Janeiro: [s.n.], 2006.

[Cancian, Stemmer e Fröhlich 2006]CANCIAN, R. L.; STEMMER, M. R.; FRöHLICH, A. A. M. Real-time embedded systems co-design. In: *In: 8th Brazilian Workshop on Real-Time Systemsl- WiP WTR*. Curitiba: [s.n.], 2006.

[Cancian, Stemmer e Fröhlich 2009]CANCIAN, R. L.; STEMMER, M. R.; FRöHLICH, A. A. M. Epos repository structure. In: *In: 8th International Information and Telecommunication Technologies Symposium*. Florianópolis: [s.n.], 2009.

[Cancian, Stemmer e Fröhlich 2009]CANCIAN, R. L.; STEMMER, M. R.; FRöHLICH, A. A. M. Implementation techniques for supporting component-based embedded systems. In: *In: 8th International Information and Telecommunication Technologies Symposium*. Florianópolis: [s.n.], 2009.

[Cancian et al. 2007]CANCIAN, R. L. et al. Ferramenta de suporte ao projeto automatizado de sistemas computacionais embarcados. In: *In: Workshop de Sistemas Operacionais*. Rio de Janeiro: [s.n.], 2007.

[Castro e Zuben 2005]Recent developments in biologically inspired computing. In: CASTRO, L. N. D.; ZUBEN, F. J. V. (Ed.). [S.l.]: Idea Group Publishings, 2005. cap. Gene expre.

[Catania et al. 2009]CATANIA, V. et al. An Effective Methodology to Multi-objective Design of Application Domain-specific Embedded Architectures. In: *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*. Ieee, 2009. p. 643–650. ISBN 978-0-7695-3782-5. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5350191>.

[Chakraborty, Mitra e Roychoudhury 2006]CHAKRABORTY, B.; MITRA, T.; ROYCHOUDHURY, a. Handling Constraints in Multi-Objective GA for Embedded System Design. *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)*, Ieee, p. 305–310, 2006. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1581469>.

[Coello 2005]COELLO, C. A. C. Evolutionary Multi-Objective Optimization: Current State and Future Challenges. *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, Ieee, p. 5–5, 2005. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1587717>.

[Coello 2006]COELLO, C. A. C. Evolutionary Multi-Objective Optimization: A Historical View of the Field. *IEEE Computational Intelligence Magazine*, v. 18, n. 1, p. 85–70, jun. 2006.

[Coello 2009]COELLO, C. A. C. Evolutionary multi-objective optimization: some current research trends and topics that remain to be explored. *Frontiers of Computer Science in China*, v. 3, p. 18–30, 2009.

[Comer 1990]COMER, E. Domain analysis: A systems approach to software reuse software productivity solutions. In: *IEEE/AIAA/NASA: Proceedings of the 9th Digital Avionics Systems Conference*. [S.l.: s.n.], 1990.

[Dawkins 1976]DAWKINS, R. *The Selfish Gene*. [S.l.]: Oxford University Press, 1976. Paperback. ISBN 019857519X.

[Dawkins 1986]DAWKINS, R. *The Blind Watchmaker*. 1st american ed. ed. New York: Norton, 1986. ISSN 0393022161.

[Deb 2001]DEB, K. *Multiobjective Optimization Using Evolutionary Algorithms*. [S.l.]: Wiley, 2001.

[Deb 2009]DEB, K. *Multi-Objective Optimization using Evolutionary Algorithms*. [S.l.]: John Wiley & Sons, LTD, 2009. ISSN 978-0-470-74361-4.

[Deb et al. 2001]DEB, K. et al. Scalable Test Problems for Evolutionary Multi-Objective Optimization. *Computer Engineering*, n. 1990, p. 1–27, 2001.

[Deb et al. 2002]DEB, K. et al. Scalable multi-objective optimization test problems. In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. Ieee, 2002. v. 2, n. i, p. 825–830. ISBN 0-7803-7282-4. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1007032>.

[Deb et al. 2006]DEB, K. et al. Evolutionary Multiobjective Optimization. In: ____. *Test*. [S.l.]: Springer Berlin Heidelberg, 2006. cap. Scalable T.

[Di Nuovo et al. 2006]Di Nuovo, A. G. et al. Fuzzy decision making in embedded system design. *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis - CODES+ISSS '06*, ACM Press, New York, New York, USA, p. 223, 2006. <http://portal.acm.org/citation.cfm?doid=1176254.1176309>.

[Ernst et al. 1993]ERNST, R. et al. Hardware/software cosynthesis for microcontrollers. In: *IEEE Design & Test of Computers*. [S.l.: s.n.], 1993. v. 10.

[Ernst e Jerraya 2000]ERNST, R.; JERRAYA, A. A. Embedded system design with multiple languages. In: *Proceedings of the 2000 conference on Asia South Pacific design automation*. Yokohama, Japan: [s.n.], 2000. p. 391–396.

[Eskesen et al. 2004]ESKESEN, S. T. et al. Periodicity of DNA in exons. *BMC molecular biology*, v. 5, p. 12, ago. 2004. ISSN 1471-2199. <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=516030&tool=pmcentrez&rendertype=abstract>.

[Falk et al. 2006]FALK, J. et al. Effcient representation and simulation of model-based designs in systemc. In: *FDL'06: Proceedings of the Forum on Specification and Design Languages 2006*. Darmstadt, Germany: [s.n.], 2006. p. 19–22.

[Ferrandi et al. 2008]FERRANDI, F. et al. A multi-objective genetic algorithm for design space exploration in high-level synthesis. In: *ISVLSI'08: Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI*. Washington, DC, USA: IEEE Computer Society, 2008. p. 417–422. ISBN 978-0-7695-3170-0.

[Ferrari e Sangiovanni-Vincetelli 1999]FERRARI, A.; SANGIOVANNI-VINCETELLI, A. System design: Traditional concepts and new paradigms. In: *Proceedings of the International Conference on Computer Design*. [S.l.: s.n.], 1999. p. 2–12.

[Ferreira 2001]FERREIRA, C. Gene Expression Programming : A New Adaptive Algorithm for Solving Problems. *Complex Systems*, v. 13, n. 2, p. 87–129, 2001.

[Ferreira 2002]FERREIRA, C. Combinatorial optimization by gene expression programming: Inversion revisited. In: SANTOS, J. M.; ZAPICO, A. (Ed.). *Proceedings of the Argentine Symposium on Artificial Intelligence*. Santa Fe, Argentina: [s.n.], 2002. p. 160–174.

[Ferreira 2002]FERREIRA, C. Discovery of the boolean functions to the best density-classification rules using gene expression programming. In: *EuroGP*. [S.l.: s.n.], 2002. p. 50–59.

[Ferreira 2002]FERREIRA, C. Genetic representation and genetic neutrality in gene expression programming. *Advances in Complex Systems*, v. 5, n. 4, p. 389–408, 2002.

[Ferreira 2002]FERREIRA, C. Mutation, transposition, and recombination: An analysis of the evolutionary dynamics. In: *JCIS*. [S.l.: s.n.], 2002. p. 614–617.

[Ferreira 2003]FERREIRA, C. Function finding and the creation of numerical constants in gene expression programming. In: SPRINGER-VERLAG (Ed.). *Advances in Soft Computing - Engineering Design and Manufacturing*. [S.l.: s.n.], 2003. p. 257–266.

[Ferreira 2006]FERREIRA, C. Automatically defined functions in gene expression programming. In: NEDJAH, N.; MOURELLE, L.; ABRAHAM, A. (Ed.). *Genetic Systems Programming*. Springer Berlin / Heidelberg,

2006, (Studies in Computational Intelligence, v. 13). p. 21–56. 10.1007/3-540-32498-4_2. <http://dx.doi.org/10.1007/3-540-32498-4_2>.

[Fonseca 1995]FONSECA, C. Multiobjective genetic algorithms made easy: selection sharing and mating restriction. *1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, Iee, n. 414, p. 45–52, 1995. <http://link.aip.org/link/IEECPS/v1995/iCP414/p45/s1&Agg=doi>.

[Fröhlich 2001]FRöHLICH, A. A. M. *GMD Research Series*. Tese (Doutorado) — Institut für Rechnerarchitektur und Softwaretechnik (FIRST), 2001.

[Fröhlich e Schrëder-Preikschat 2000]FRöHLICH, A. A. M.; SCHRëDER-PREIKSCHAT, W. Scenario adapters: Efficiently adapting components. In: *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics*. Orlando, USA: [s.n.], 2000. <http://www.lisha.ufsc.br/ guto/publications/sci2000.pdf>.

[Goldbarg, Bagi e Goldbarg 2008]GOLDBARG, M. C.; BAGI, L. B.; GOLDBARG, E. F. G. Algoritmo transgenético aplicado ao problema do caixeiro comprador capacitado simétrico. *Pesquisa Operacional*, scielo, v. 28, p. 93 – 121, 04 2008. ISSN 0101-7438. <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382008000100006&nrm=iso>.

[Gras 2008]GRAS, R. How efficient are genetic algorithms to solve high epistasis deceptive problems? In: *Evolutionary Computation*. [S.l.: s.n.], 2008. p. 242–249.

[Group 2011]GROUP, O. M. *MDA*. May 2011. Http://www.omg.org/mda. <http://www.omg.org/mda>.

[Han et al. 2007]HAN, M. et al. Design space exploration in multi-objective hierarchical soc design. In: *ASICON'07: Proceedings of the 7th IEEE International Conference on ASIC*. [S.l.]: IEEE Computer Society, 2007. p. 118–121.

[Handoko, Kwoh e Ong 2010]HANDOKO, S.; KWOH, C.; ONG, Y. Feasibility structure modeling: an effective chaperone for

constrained memetic algorithms. *Evolutionary Computation, IEEE Transactions on*, IEEE, v. 14, n. 5, p. 740–758, 2010. <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5570981>.

[Haubelt et al. 2008]HAUBELT, C. et al. Systemcodesigner: automatic design space exploration and rapid prototyping from behavioral models. In: *DAC '08: Proceedings of the 45th annual Design Automation Conference*. New York, NY, USA: ACM, 2008. p. 580–585. ISBN 978-1-60558-115-6.

[Haubelt e Teich 2003]HAUBELT, C.; TEICH, J. Accelerating design space exploration using pareto-front arithmetics. In: *ASP-DAC'03: Proceedings of the 2003 Asia and South Pacific Design Automation Conference*. New York, NY, USA: ACM, 2003. p. 525–531. ISBN 0-7803-7660-9.

[Hermsen, Ursem e Wolde 2010]HERMSEN, R.; URSEM, B.; WOLDE, P. R. ten. Combinatorial gene regulation using auto-regulation. *PLoS computational biology*, v. 6, n. 6, p. e1000813, jun. 2010. ISSN 1553-7358. <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2883594&tool=pmcentrez&rendertype=abstract>.

[Horowitz et al. 2003]HOROWITZ, B. et al. Platform-based embedded software design and system integration for autonomous vehicles. *Proceedings of the IEEE*, v. 91, n. 1, p. 198–211, jan. 2003. ISSN 0018-9219. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1173213>.

[Huband et al. 2006]HUBAND, S. et al. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, v. 10, n. 5, p. 477–506, out. 2006. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1705400>.

[Ikeda, Kita e Kobayashi 2001]IKEDA, K.; KITA, H.; KOBAYASHI, S. Failure of Pareto-based MOEAs: does non-dominated really mean near to optimal? In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. Ieee, 2001. p. 957–962. ISBN 0-7803-6657-3. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=934293>.

[Inc 2010]INC, S. *Synopsys Predictable Success*. [S.l.], Mai 2010.

[Initiative 2011]INITIATIVE, O. S. *SystemC Documentation*. [S.l.], Mai 2011.

[Ishibuchi et al. 2009]ISHIBUCHI, H. et al. Evolutionary many-objective optimization by NSGA-II and MOEA/D with large populations. *2009 IEEE International Conference on Systems, Man and Cybernetics*, Ieee, v. 1, n. October, p. 1758–1763, out. 2009. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5346628>.

[Ishibuchi et al. 2009]ISHIBUCHI, H. et al. Selecting a small number of representative non-dominated solutions by a hypervolume-based solution selection approach. In: *2009 IEEE International Conference on Fuzzy Systems*. Ieee, 2009. p. 1609–1614. ISBN 978-1-4244-3596-8. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5277324>.

[Ishibuchi, Tsukamoto e Nojima 2007]ISHIBUCHI, H.; TSUKAMOTO, N.; NOJIMA, Y. Iterative Approach to Indicator-Based Multiobjective Optimization. In: *Evolutionary Computation*. [S.l.: s.n.], 2007. p. 3967–3974.

[Ishibuchi, Tsukamoto e Nojima 2008]ISHIBUCHI, H.; TSUKAMOTO, N.; NOJIMA, Y. Evolutionary Many-Objective Optimization: A Short Review. In: *Evolutionary Computation*. [S.l.: s.n.], 2008. p. 2419–2426.

[Jerraya e Wolf 2005]JERRAYA, A.; WOLF, W. Hardware/software interface codesign for embedded systems. In: . [S.l.: s.n.], 2005. v. 38, p. 63–69. Digital Object Identifier: 10.1109/MC.2005.61.

[Keutzer et al. 2000]KEUTZER, K. et al. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 19, n. 12, p. 1523–1543, 2000. ISSN 02780070. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=898830>.

[Knowles e Corne 2002]KNOWLES, J. D.; CORNE, D. On metrics for comparing nondominated sets. In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02*

(Cat. No.02TH8600). Ieee, 2002. p. 711–716. ISBN 0-7803-7282-4. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1007013>.

[Krasnogor e Smith 2005]KRASNOGOR, N.; SMITH, J. A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues. *IEEE Transactions on Evolutionary Computation*, v. 9, n. 5, p. 474–488, out. 2005. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1514472>.

[Ku, Thiele e Zitzler 2005]KU, S.; THIELE, L.; ZITZLER, E. Modular design space exploration framework for embedded systems. *Computer Engineering*, v. 152, n. 2, p. 183–192, 2005.

[Kuhn e Rosenstiel 2000]KUHN, T.; ROSENSTIEL, W. Java based object oriented hardware specification and synthesis. In: *ASP-DAC '00: Proceedings of the Asia and South Pacific Design Automation Conference*. [S.l.: s.n.], 2000. p. 579–584.

[Kukkonen, Member e Lampinen 2007]KUKKONEN, S.; MEMBER, S.; LAMPINEN, J. Ranking-Dominance and Many-Objective Optimization. *2007 IEEE Congress on Evolutionary Computation*, Ieee, p. 3983–3990, set. 2007. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4424990>.

[Lara et al. 2010]LARA, A. et al. HCS: A New Local Search Strategy for Memetic Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, v. 14, n. 1, p. 112–132, fev. 2010. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5352350>.

[Laumanns et al. 2002]LAUMANNS, M. et al. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, v. 10, n. 3, p. 263–82, jan. 2002. ISSN 1063-6560. <http://www.ncbi.nlm.nih.gov/pubmed/12227996>.

[Le e Landa-Silva 2007]LE, K.; LANDA-SILVA, D. Obtaining Better Non-Dominated Sets Using Volume Dominance. In: *2007 IEEE Congress*

on Evolutionary Computation. Ieee, 2007. D, p. 3119–3126. ISBN 978-1-4244-1339-3. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4424870>.

[Lee e El-Sharkwaki 2008]LEE, K. Y.; EL-SHARKWAKI, M. A. *Modern Heuristic Optimization Techniques: Theory and Applications to Power Systems*. [S.l.]: John Wiley & Sons, 2008.

[Lewin 2007]LEWIN, B. *Genes*. 9th edition. ed. [S.l.]: Jones & Bartlett Publishers, 2007. ISSN 978-0763740634.

[Li et al. 2009]LI, K. et al. A novel algorithm for non-dominated hypervolume-based multiobjective optimization. In: *2009 IEEE International Conference on Systems, Man and Cybernetics*. Ieee, 2009. p. 5220–5226. ISBN 978-1-4244-2793-2. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5345983>.

[Li et al. 2005]LI, X. et al. Prefix gene expression programming. In: ROTHLAUF, F. (Ed.). *GECCO '05: Late breaking paper at Genetic and Evolutionary Computation Conference*. Washington, D.C., USA: [s.n.], 2005. <http://www.cs.bham.ac.uk/ wbl/biblio/gecco2005lbp/papers/85-li.pdf>.

[Lodish et al. 2000]LODISH, H. et al. *Molecular Cell Biology*. fourth edition. [S.l.]: W. H. Freeman and Company, 2000.

[Lukasiewycz, Glaß e Reimann 2010]LUKASIEWYCZ, M.; GLAß, M.; REIMANN, F. *Meta-heuristic Optimization Framework for Java*. October 2010. Http://opt4j.sourceforge.net/. <http://opt4j.sourceforge.net/>.

[Marcondes et al. 2009]MARCONDES, H. et al. Modelagem e implementação de escalonadores de tempo real para sistemas embarcados. In: *In: VI Workshop de Sistemas Operacionais*. Bento Gonçalves: [s.n.], 2009.

[Marcondes et al. 2009]MARCONDES, H. et al. On the design of flexible real-time schedulers for embedded systems. In: *In: Symposium on Embedded and Pervasive Systems (EPS-09)*. Vancouver, Canada: [s.n.], 2009.

[Marcondes et al. 2006]MARCONDES, H. et al. EPOS : Um Sistema Operacional Portável para Sistemas Profundamente Embarcados. p. 31–45, 2006.

[Mei, Schaumont e Vernalde 2000]MEI, B.; SCHAUMONT, P.; VERNALDE, P. Hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems. In: *Proceedings of the ProRISC2000*. [S.l.: s.n.], 2000.

[Micheli 1994]MICHELI, G. D. *Synthesis and Optimization of Digital Circuits*. [S.l.]: McGraw-Hill Higher Education, 1994. ISBN 0070163332.

[Micheli 1999]MICHELI, G. de. Hardware synthesis from c/c++ models. In: *DATE '99: Proceedings of the Design, Automation and Test in Europe*. [S.l.: s.n.], 1999.

[Miettinen 1999]MIETTINEN, K. *Nonlinear Multiobjective Optimization*. Boston: Kluwer, 1999.

[Mohanty et al. 2002]MOHANTY, S. et al. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In: *LCTES/SCOPES '02: Proceedings of the joint conference on Languages, compilers and tools for embedded systems*. New York, NY, USA: ACM, 2002. p. 18–27. ISBN 1-58113-527-0.

[Muller-Glaser et al. 2004]MULLER-GLASER, K. et al. Multiparadigm modeling in embedded systems design. In: *IEEE Transactions on Control System Technology*. [S.l.: s.n.], 2004. v. 12-2, p. 279–292.

[Noonan e Flanagan 2006]NOONAN, L.; FLANAGAN, C. Utilising evolutionary approaches and object oriented techniques for design space exploration. In: *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*. Washington, DC, USA: IEEE Computer Society, 2006. p. 346–352. ISBN 0-7695-2609-8.

[Novozhilov, Wolf e Koonin 2007]NOVOZHILOV, A. S.; WOLF, Y. I.; KOONIN, E. V. Evolution of the genetic code: partial optimization of a random code for robustness to translation error in a rugged fitness landscape. *Biology direct*, v. 2, p. 24, jan. 2007. ISSN 1745-6150. <http://www.ncbi.nlm.nih.gov/pubmed/17956616>.

[Oyamada et al. 2007]OYAMADA, M. et al. Software Performance Estimation in MPSoC Design. *2007 Asia and South Pacific Design Automation Conference*, Ieee, p. 38–43, jan.

2007. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4195993>.

[Palermo, Silvano e Zaccaria 2008]PALERMO, G.; SILVANO, C.; ZACCARIA, V. An efficient design space exploration methodology for on-chip multiprocessors subject to application-specific constraints. *Application Specific Processors, Symposium on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 75–82, 2008.

[Palesi e Givargis 2002]PALESI, M.; GIVARGIS, T. Multi-objective design space exploration using genetic algorithms. In: *CODES '02: Proceedings of the 10th international symposium on Hardware/software codesign*. New York, NY, USA: ACM, 2002. p. 67–72. ISBN 1-58113-542-4.

[Palma et al. 2005]PALMA, J. C. S. et al. Mapping embedded systems onto nocs: the traffic effect on dynamic energy estimation. In: *SBCCI '05: Proceedings of the 18th Symposium on Integrated Circuits and Systems Design*. [S.l.]: ACM, 2005.

[Parnas 1976]PARNAS, D. L. On the design and development of program families. *IEEE Transactions on Software Engineering*, SE-2, n. 1, p. 1–9, mar 1976.

[Parreiras e Vasconcelos 2009]PARREIRAS, R.; VASCONCELOS, J. a. A. Decision making in multiobjective optimization aided by the multicriteria tournament decision method. *Nonlinear Analysis: Theory, Methods & Applications*, Elsevier Ltd, v. 71, n. 12, p. e191–e198, dez. 2009. ISSN 0362546X. <http://linkinghub.elsevier.com/retrieve/pii/S0362546X08005440>.

[Parreiras, Maciel e Vasconcelos 2006]PARREIRAS, R. O.; MACIEL, J. a. H. R. D.; VASCONCELOS, J. a. A. The A Posteriori Decision in Multiobjective Optimization Problems With Smarts , Promethee II , and a Fuzzy Algorithm. *Optimization*, v. 42, n. 4, p. 1139–1142, 2006.

[Pereira e Cancian 2008]PEREIRA, M. C.; CANCIAN, R. L. Implementação de componentes de sistemas operacionais embarcados em hardware. In: *In: Congresso Sul Brasileiro de Computação*. Bento Gonçalves: [s.n.], 2008.

[Pietro-Diaz 1990]PIETRO-DIAZ, R. *Domain Analysis - An Introdution*. [S.l.], 1990.

[Pilato et al. 2008]PILATO, C. et al. High-level synthesis with multi-objective genetic algorithm: A comparative encoding analysis. *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, Ieee, p. 3334–3341, jun. 2008. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4631249>.

[Polpeta e Fröhlich 2004]POLPETA, F. V.; FRöHLICH, A. A. M. Hardware mediators: a portability artifact for component-based systems. In: *Proceedings of the International Conference on Embedded and Ubiquitous Computing*. Aizu, Japan: Springer, 2004. (Lecture Notes in Computer Science, v. 3207), p. 271–280. <http://www.lisha.ufsc.br/ guto/publications/euc2004a.pdf>.

[Purshouse e Fleming 2007]PURSHOUSE, R. C.; FLEMING, P. J. On the Evolutionary Optimization of Many Conflicting Objectives. *IEEE Transactions on Evolutionary Computation*, v. 11, n. 6, p. 770–784, dez. 2007. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4384508>.

[Rachmawati e Srinivasan 2009]RACHMAWATI, L.; SRINIVASAN, D. Multiobjective Evolutionary Algorithm With Controllable Focus on the Knees of the Pareto Front. *IEEE Transactions on Evolutionary Computation*, v. 13, n. 4, p. 810–824, ago. 2009. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5208606>.

[Said, Bechikh e Ghédira 2010]SAID, L. B.; BECHIKH, S.; GHéDIRA, K. The r-Dominance: A New Dominance Relation for Interactive Evolutionary Multicriteria Decision Making. *IEEE Transactions on Evolutionary Computation*, p. 1–18, 2010. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5453088>.

[Sangiovanni-Vincentelli 2005]SANGIOVANNI-VINCENTELLI, a. System-level design: a strategic investment for the future of the electronic industry. *2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT).*, Ieee, p. 1–5, 2005. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1500004>.

[Sangiovanni-Vincentelli 2007]SANGIOVANNI-VINCENTELLI, A. Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design. *Proceedings of the IEEE*, v. 95, n. 3, p. 467–506, mar. 2007. ISSN 0018-9219. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4167779>.

[Sangiovanni-Vincentelli e Martin 2001]SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. *Design Test of Computers, IEEE*, v. 18, n. 6, p. 23 –33, nov. 2001. ISSN 0740-7475.

[Sangiovanni-vincentelli e Martin 2001]SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, v. 18, n. 6, p. 23–33, 2001. ISSN 07407475. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=970421>.

[Sangiovanni-Vincentelli et al. 2009]SANGIOVANNI-VINCENTELLI, A. et al. Metamodeling: An Emerging Representation Paradigm for System-Level Design. *IEEE Design & Test of Computers*, v. 26, n. 3, p. 54–69, maio 2009. ISSN 0740-7475. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5167508>.

[Santos, Cancian e Fröhlich 2006]SANTOS, D. M.; CANCIAN, R. L.; FRöHLICH, A. A. M. Desenvolvimento de sistemas embarcados com suporte a tempo-real seguindo o projeto de sistemas orientados a aplicação. In: *8th Brazilian Workshop on Real-Time Systems - WiP WTR*. Curitiba: [s.n.], 2006.

[Santos et al. 2006]SANTOS, D. M. et al. Advantages and disadvantages of application-oriented system design in embedded systems design. In: *Proceedings of 4th International IEEE Conference on Industrial Informatics*. Cingapura: [s.n.], 2006. p. 904–909.

[Santos et al. 2006]SANTOS, D. M. et al. Application-oriented system design as an embedded systems development strategy: a critical analisys. In: *ETFA '06: Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*. Prague: [s.n.], 2006.

[Schirrmeister e Sangiovanni-Vincentelli 2001]SCHIRRMEISTER, F.; SANGIOVANNI-VINCENTELLI, A. Virtual component co-design-applying function architecture co-design to automotive applications. In: *IVEC '01: Proceedings of the IEEE International Vehicle Electronics Conference*. [S.l.: s.n.], 2001. p. 221–226.

[Schlichter, Haubelt e Teich 2005]SCHLICHTER, T.; HAUBELT, C.; TEICH, J. Improving ea-based design space exploration by utilizing symbolic feasibility tests. In: *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005. p. 1945–1952. ISBN 1-59593-010-8.

[Schlichter et al. 2006]SCHLICHTER, T. et al. Improving system level design space exploration by incorporating sat-solvers into multi-objective evolutionary algorithms. In: *ISVLS I'06: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*. Washington, DC, USA: IEEE Computer Society, 2006. p. 309–315. ISBN 0-7695-2533-4.

[Schulter et al. 2007]SCHULTER, A. et al. A tool for supporting and automating the development of component-based embedded systems. *Journal of Object Technology*, v. 6, p. 399–416, 2007.

[Smith, Bolton e Nguyen 2010]SMITH, C. L.; BOLTON, A.; NGUYEN, G. Genomic and epigenomic instability, fragile sites, schizophrenia and autism. *Current genomics*, v. 11, n. 6, p. 447–69, set. 2010. ISSN 1875-5488. <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3018726&tool=pmcentrez&rendertype=abstract>.

[Smith 2007]SMITH, J. E. Coevolving memetic algorithms: a review and progress report. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, v. 37, n. 1, p. 6–17, fev. 2007. ISSN 1083-4419. <http://www.ncbi.nlm.nih.gov/pubmed/17278554>.

[So, Diniz e Hall 2003]SO, B.; DINIZ, P.; HALL, M. Using estimates from behavioral synthesis tools in compiler-directed design space exploration. In: *DAC '03: Procedings of the 40th Conference on Desig Automation*. [S.l.]: ACM Press, 2003.

[Streubühr et al. 2006]STREUBüHR, M. et al. Task-accurate performance modeling in systemc for real-time multi-processor architectures. In: *DATE'06: Proceedings of the conference on Design, automation and test in Europe*. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006. p. 480–481. ISBN 3-9810801-0-6.

[Systems 2009]SYSTEMS, F. D. *Forte Design Systems: Market and technology leader for ASIC & SoC high-level design | Cynthesizer: The Industry Leader in ESL Synthesis*. Mai 2009. Http://www.forteds.com. <http://www.forteds.com>.

[Thiele et al. 2001]THIELE, L. et al. Funstate- an internal design representation for codesign. In: *VLSI '01: Proceedings of the IEEE Transactions on Very Large Scale Integration*. [S.l.: s.n.], 2001.

[Thomas, Adams e Schmit 1993]THOMAS, D. E.; ADAMS, J. K.; SCHMIT, H. A model and methodology for hardware-software codesign. In: *IEEE Design & Test of Computers*. [S.l.: s.n.], 1993. v. 10, p. 6–15.

[TIK 2008]TIK, E. Z. I. *ETH - SOP - Download/Material - Supplementary Material - Testproblems*. 2008. Http://www.tik.ee.ethz.ch/ sop/download/supplementary/testproblems/. <http://www.tik.ee.ethz.ch/ sop/download/supplementary/testproblems/>.

[Tondello e Fröhlich 2004]TONDELLO, G. F.; FRöHLICH, A. A. M. Configuration Management of Embedded Operating Systems using Application-Oriented System Design. *World Wide Web Internet And Web Information Systems*, 2004.

[Tondello e Fröhlich 2005]TONDELLO, G. F.; FRöHLICH, A. A. M. On the automatic configuration of application-oriented operating systems. *The 3rd ACS/IEEE International Conference on-Computer Systems and Applications, 2005.*, Ieee, p. 637–640, 2005. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1387109>.

[Tran 2006]TRAN, K. D. *An Improved Multi-Objective Evolutionary Algorithm with Adaptable Parameters*. Tese (Doutorado) — Graduate School of Computer and Information Systems, Nova Southeastern University, August 2006.

[Tung et al. 2010]TUNG, Y. et al. Platform-based design automation-Platform Core Compiler. In: *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*. IEEE, 2010. p. 33–35. <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5496685>.

[Wang e Tai 2007]WANG, N.; TAI, K. Handling objectives as adaptive constraints for multiobjective structural optimization. *2007 IEEE Congress on Evolutionary Computation*, Ieee, v. 639798, p. 3922–3929, set. 2007. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4424982>.

[Wang, Kodase e Shin 2002]WANG, S.; KODASE, S.; SHIN, G. Automating embedded software construction and analysis with design models. In: *Proceeding of International Conference Euro-uRapid*. [S.l.: s.n.], 2002.

[Wanner et al. 2008]WANNER, E. F. et al. Multiobjective Memetic Algorithms With Quadratic Approximation-Based Local Search for Expensive Optimization in Electromagnetics. *IEEE Transactions on Magnetics*, v. 44, n. 6, p. 1126–1129, jun. 2008. ISSN 0018-9464. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4527023>.

[Wolf 2001]WOLF, W. *Computers as Components - Principles of Embedded Computing System Design*. 1. ed. San Francisco: Morgan Kaufmann Publishers, 2001. 662 p.

[Xie e Ding 2009]XIE, C.; DING, L. Selection Strategies of Evolutionary Algorithms in Multiobjective Optimization. *2009 Fifth International Conference on Natural Computation*, Ieee, p. 633–637, ago. 2009. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5363435>.

[Zhang et al. 2009]ZHANG, J. et al. An improved multi-objective genetic algorithm based on pareto front and fixed point theory. In: *ISA '09: International Workshop on Intelligent Systems and Applications*. [S.l.: s.n.], 2009. p. 1–5.

[Zielinski e Rutkowski 2006]ZIELINSKI, L.; RUTKOWSKI, J. Applied Soft Computing Technologies: The Challenge of Complexity. In: ____. Berlin: Springer Berlin / Heidelberg, 2006. v. 2, n. 4, cap. Design Cen, p. 91–98.

[Zitzler 1999]ZITZLER, E. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Tese (Doutorado) — ETH Zurich, Switzerland, 1999.

[Zitzler, Brockhoff e Thiele 2007]ZITZLER, E.; BROCKHOFF, D.; THIELE, L. The Hypervolume Indicator Revisited : On the Design of Pareto-compliant Indicators Via. p. 862–876, 2007.

[Zitzler et al. 1998]ZITZLER, E. et al. Why Quality Assessment of Multiobjective Optimizers Is Difficult. *Scenario*, 1998.

[Zitzler e Thiele 1999]ZITZLER, E.; THIELE, L. *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strenght Pareto Approach*. 1999.

[Zitzler e Thiele 1999]ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, v. 3, n. 4, p. 257–271, 1999. ISSN 1089778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=797969>.

[Zitzler et al. 2003]ZITZLER, E. et al. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, v. 7, n. 2, p. 117–132, abr. 2003. ISSN 1089-778X. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1197687>.

[Zou et al. 2008]ZOU, X. et al. A new evolutionary algorithm for solving many-objective optimization problems. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, v. 38, n. 5, p. 1402–12, out. 2008. ISSN 1941-0492. <http://www.ncbi.nlm.nih.gov/pubmed/19109642>.

[Zou et al. 2008]ZOU, X. et al. A new evolutionary algorithm for solving many-objective optimization problems. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, v. 38, n. 5, p. 1402–12, out. 2008. ISSN 1941-0492. <http://www.ncbi.nlm.nih.gov/pubmed/19109642>.