

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

**Uma Abordagem para Recomendação de Módulos para
Projetos de Desenvolvimento Distribuído de
Linhas de Produto de Software**

Thaís Alves Burity Pereira

João Pessoa - PB

Fevereiro de 2011

Thaís Alves Burity Pereira

**Uma Abordagem para Recomendação de Módulos para
Projetos de Desenvolvimento Distribuído de
Linhas de Produto de Software**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como requisito parcial para obtenção do título de Mestre em Informática (Sistemas de Computação).

Orientador: Prof. Dr. Glêdson Elias da Silveira

João Pessoa - PB

Fevereiro de 2011

P436u Pereira, Thaís Alves Burity.

Uma abordagem para recomendação de módulos para projetos de desenvolvimento distribuído de linhas de produto de software / Thaís Alves Burity Pereira.- João Pessoa, 2011.
115f.

Orientador: Gledson Elias da Silveira

Dissertação (Mestrado) - UFPB/CCEN

1. Sistemas de Computação. 2. Linhas de Produto de Software (LPS). 3. Desenvolvimento Distribuído de Software (DDS). 4. Métricas arquiteturais. Metaheurísticas.

UFPB/BC

CDU: 004(043)

1

Ata da Sessão Pública de Defesa de Dissertação de
Mestrado do THAIS ALVES BURITY PEREIRA,
candidata ao Título de Mestre em Informática na
Área de Sistemas de Computação, realizada em 28
de fevereiro de 2011.

2
3

6 Aos vinte e oito dias do mês de fevereiro do ano dois mil e onze, às quatorze horas, na Sala
7 de Reunião do Centro de Ciências Exatas e da Natureza da Universidade Federal da
8 Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar a
9 candidata ao grau de Mestre em Informática, na área de "Sistemas de Computação", na
10 linha de pesquisa "Computação Distribuída", o Sra. THAIS ALVES BURITY PEREIRA.
11 A comissão examinadora composta pelos professores doutores: Glêdson Elias da Silveira
12 (DI - UFPB), Primeiro Orientador e Presidente da Banca Examinadora, Tatiana Aires
13 Tavares (DI-UFPB) como examinador interno e Elisa Hatsue Moriya Huzita (UEM) como
14 examinador externo. Dando início aos trabalhos, a Profª. Tatiana Aires Tavares
15 cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a
16 palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de
17 dissertação intitulado "Uma Abordagem para Recomendação de Módulos para Projetos de
18 Desenvolvimento Distribuído de Linhas de Produto de Software". Concluída a exposição, a
19 candidata foi argüida pela Banca Examinadora que emitiu o seguinte parecer: "aprovado".
20 Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de
21 Mestre em Informática na forma da lei e, para constar, a professora Tatiana Aires Tavares,
22 Sra. Coordenadora do PPGI, lavrou a presente ata, que vai assinada por ela, e pelos
23 membros da Banca Examinadora. João Pessoa, 28 de fevereiro de 2011.

24


Tatiana Aires Tavares

25

26

Prof. Dr. Glêdson Elias da Silveira
Primeiro Orientador (DI-UFPB)

Profa. Dra. Tatiana Aires Tavares
Examinador Interno (DI-UFPB)

Profa. Dra. Elisa Hatsue Moriya Huzita
Examinador Externo (UEM)

27


Tatiana Aires Tavares


Resumo

Linhas de Produto de Software (LPS) tem ganhado bastante espaço na indústria de software nos últimos anos, principalmente por promover o reuso de maneira sistemática e previsível, e oferecer apoio ao desenvolvimento de produtos para mercados globais. No entanto, o desenvolvimento de LPS exige das organizações grande investimento inicial e a participação de profissionais qualificados, os quais nem sempre estão disponíveis localmente. Nesse cenário, o Desenvolvimento Distribuído de Software (DDS) pode ser empregado para encontrar especialistas do domínio e equipes mais qualificadas para o desenvolvimento de LPS. Além disso, essa forma de trabalho reforça alguns dos benefícios já oferecidos por LPS, tais como redução de custo de desenvolvimento e aumento da qualidade dos produtos. Apesar disso, abordagens de DDS também têm suas limitações, relacionadas principalmente à comunicação entre as equipes participantes de um mesmo projeto. Partindo da premissa de que dependências entre componentes de software exercem grande influência sobre a necessidade de comunicação entre suas respectivas equipes de desenvolvimento, neste trabalho é apresentada uma abordagem para identificar candidatos a módulos para serem desenvolvidos de forma (parcialmente) independente por equipes geograficamente dispersas, sendo um módulo um agrupamento de componentes. Para tanto, a abordagem define: medidas quantitativas que descrevem a dependência entre componentes de software em projetos de LPS; um algoritmo baseado em metaheurísticas para agrupar componentes em módulos, tratando o agrupamento como um problema de otimização; e uma medida quantitativa que descreve as dependências entre módulos, que deve ser empregada para guiar a alocação das equipes de desenvolvimento aos módulos.

Palavras-chave: Linhas de Produto de Software, Desenvolvimento Distribuído de Software, Métricas Arquiteturais, Metaheurísticas, Engenharia de Software Baseada em Buscas.

Abstract

Software Product Line (SPL) has been adopted by software industry in recent years, mainly by promoting software reuse in a systematic and predictable way, and supporting product development for global markets. Despite the benefits, SPL requires a high initial effort and the involvement of domain experts, which are not always available in a local team. In such a scenario, Global Software Development (GSD) approaches would be applied to find domain experts and more qualified teams for SPL projects. Moreover, such work strategy reinforces some of the benefits already offered by SPL approach, such as reducing development cost and increasing product quality. Nonetheless, GSD approaches also present some obstacles, which are mainly related to communication between dispersed development teams. Assuming that dependencies between software components greatly influence the need for communication between their respective development teams, in this work it is presented an approach to identify candidates for modules to be developed in a (partially) independent manner by geographically dispersed teams, in which a module is a clustering of components. To do so, the approach defines: quantitative measures that describe the dependence between software components in SPL projects; an algorithm based on metaheuristics for clustering components into modules, dealing with clustering as an optimization problem; and a quantitative measure that describe the dependencies between modules, which must be employed to guide the allocation of the development teams to the modules.

Keywords: Software Product Lines, Global Software Development, Architectural Metrics, Metaheuristics, Search Based Software Engineering.

Sumário

RESUMO.....	III
ABSTRACT.....	VI
SUMÁRIO	VII
LISTA DE FIGURAS.....	IX
LISTA DE TABELAS	XI
LISTA DE SIGLAS	XII
1. INTRODUÇÃO.....	1
1.1. Motivação	5
1.2. Objetivos	7
1.3. Organização do Trabalho	8
2. FUNDAMENTAÇÃO TEÓRICA.....	9
2.1. Linhas de Produto de Software.....	9
2.2. Arquitetura de Software.....	14
2.3. <i>Design Structure Matrix</i>	19
2.4. Engenharia de Software Baseada em Buscas	21
3. TRABALHOS RELACIONADOS.....	25
3.1. Agrupamento de Componentes na Engenharia de Domínio	25
3.2. Agrupamento de Componentes em DDS	27
3.3. Agrupamento de Componentes Aplicado à Engenharia Reversa	28
3.3.1. Função Objetivo	29
3.3.2. Estratégia de Busca	33
3.4. Agrupamento de Elementos Aplicado à Engenharia Mecânica	34
3.5. Considerações Finais	38
4. FRAMEWORK DE RECOMENDAÇÃO PARA ALOCAÇÃO DE EQUIPES EM PROJETOS DISTRIBUÍDOS DE LPS ...	39
4.1. Introdução.....	39
4.2. Fluxo de Atividades do Framework.....	40
4.2.1. Análise da Arquitetura.....	41
4.2.2. Avaliação Técnica	42
4.2.3. Avaliação Não-Técnica.....	43
4.2.4. Avaliação do Desempenho das Equipes	44
4.3. Considerações Finais	45
5. ABORDAGEM PARA RECOMENDAÇÃO DE MÓDULOS PARA PROJETOS DISTRIBUÍDOS DE LPS	46
5.1. Medição de Dependências entre Componentes	48

5.1.1. Redução de Perspectiva	51
5.1.2. Construção da DSM	51
5.2. Recomendação de Módulos	53
5.2.1. Premissas	55
5.2.2. Função Objetivo	57
5.2.3. Funcionamento Geral do Algoritmo.....	58
5.3. Medição de Dependências entre Módulos	62
5.4. Considerações Finais	63
6. ESTUDOS DE CASO	65
6.1. Linha de Produto de Microondas.....	66
6.1.1. Caso Baseado nas Métricas DI e DF	71
6.1.2. Caso Baseado nas Métricas DO e DF	75
6.2. Linha de Produto de Comércio Eletrônico.....	80
6.2.1. Primeira Iteração de Desenvolvimento	81
6.2.2. Segunda Iteração de Desenvolvimento.....	89
6.3. Multi-MOM	93
6.4. Gíngua-CC	101
6.5. Considerações Finais	106
7. CONSIDERAÇÕES FINAIS.....	110
7.1. Contribuições	111
7.2. Limitações	112
7.3. Trabalhos Futuros.....	113
REFERÊNCIAS	115

Lista de Figuras

Figura 1. Exemplo de diagrama de <i>features</i>	12
Figura 2. Exemplo de DSM.....	20
Figura 3. Fluxo de atividades do <i>framework</i>	41
Figura 4. Organização da abordagem proposta.	46
Figura 5. Exemplo de DSM para um projeto.	47
Figura 6. Exemplo de CMM para um projeto.	47
Figura 7. Exemplo de MBM para um projeto.	48
Figura 8. Fluxograma do algoritmo baseado em SA.	59
Figura 9. Arquitetura da LPS de microondas [Gomaa 2004].....	67
Figura 10. Componente <i>Microwave Display</i> [Gomaa 2004].	68
Figura 11. Visão detalhada dos componentes parametrizáveis [Gomaa 2004].....	69
Figura 12. DSM da LPS de microondas segundo a métrica DF.....	71
Figura 13. DSM da LPS de microondas segundo a métrica DI.....	72
Figura 14. DSM da LPS de microondas segundo as métricas DI e DF.	72
Figura 15. CMM da solução encontrada para a LPS de microondas.....	73
Figura 16. DSM da LPS de microondas reordenada.....	73
Figura 17. CMM da solução alternativa para a LPS de microondas.	74
Figura 18. DSM da LPS de microondas reordenada de acordo com a solução alternativa.	75
Figura 19. MBM da LPS de microondas segundo as métricas DI e DF.....	75
Figura 20. DSM da LPS de microondas segundo a métrica DO.	79
Figura 21. DSM da LPS de microondas segundo as métricas DO e DF.....	80
Figura 22. MBM da LPS de microondas segundo as métricas DO e DF.	80
Figura 23. Arquitetura de domínio da LPS de E-Commerce [Gomaa 2004].....	81
Figura 24. Visão detalhada dos componentes variantes [Gomaa 2004].	82
Figura 25. DSM da primeira iteração da LPS de E-Commerce.	84
Figura 26. CMM da primeira iteração da LPS de E-Commerce.	84
Figura 27. DSM da primeira iteração da LPS de E-Commerce reordenada.	85
Figura 28. DSM da primeira solução alternativa para a LPS de E-Commerce reordenada.	86
Figura 29. DSM da segunda solução alternativa para a LPS de E-Commerce reordenada.....	87
Figura 30. MBM da primeira iteração da LPS de E-Commerce.	88

Figura 31. MBM da primeira solução alternativa para a LPS de E-Commerce.....	88
Figura 32. MBM da segunda solução alternativa para a LPS de E-Commerce.	89
Figura 33. Arquitetura da aplicação B2B [Gomaa 2004].	90
Figura 34. Arquitetura da aplicação B2C [Gomaa 2004].	90
Figura 35. Componentes opcionais da LPS de E-Commerce [Gomaa 2004].	91
Figura 36. DSM da segunda iteração da LPS de E-Commerce.....	91
Figura 37. DSM que descreve a solução da segunda iteração da LPS de E-Commerce.	92
Figura 38. MBM da segunda iteração de desenvolvimento.	93
Figura 39. Arquitetura de domínio do Multi-MOM.	94
Figura 40. DSM do Multi-MOM.	95
Figura 41. CMM da solução encontrada para o Multi-MOM.	95
Figura 42. DSM do Multi-MOM reordenada de acordo com a solução encontrada.	96
Figura 43. CMM da solução do arquiteto.	96
Figura 44. DSM do Multi-MOM reordenada de acordo com a solução do arquiteto.....	97
Figura 45. DSM do Multi-MOM corrigida.	98
Figura 46. CMM da nova solução encontrada para o Multi-MOM.....	99
Figura 47. DSM do Multi-MOM reordenada segundo a nova solução encontrada.....	99
Figura 48. DSM do Multi-MOM reordenada de acordo com a solução do arquiteto.....	99
Figura 49. MBM das soluções definidas para o Multi-MOM.	100
Figura 50. Diagrama de componentes do Ginga-CC [Souza Filho et al. 2007].....	102
Figura 51. DSM do Ginga-CC.....	103
Figura 52. CMM do Ginga-CC.....	103
Figura 53. DSM do Ginga-CC reordenada de acordo com a solução encontrada.....	104
Figura 54. CMM da solução definida pelo arquiteto.	104
Figura 55. DSM do Ginga-CC reordenada de acordo com a solução do arquiteto.	105
Figura 56. MBM das soluções definidas para o Ginga-CC.....	106

Lista de Tabelas

Tabela 1. Descrição genérica de um processo de LPS.....	11
Tabela 2. Métricas de modularidade baseada em conceitos.....	17
Tabela 3. Problemas de busca em engenharia de software.	22
Tabela 4. Algoritmos para agrupamento aplicados à engenharia reversa.	29
Tabela 5. Algoritmos para agrupamento aplicados à engenharia mecânica.	35
Tabela 6. Valores adotados para os parâmetros do algoritmo baseado em SA.	66
Tabela 7. Relação de <i>features</i> por componente na LPS de microondas.	69
Tabela 8. Interfaces requeridas pelos componentes da LPS de microondas.	76
Tabela 9. Operações providas pelos componentes da LPS de microondas.....	77
Tabela 10. Relação de <i>features</i> por componente na LPS de E-Commerce.....	83
Tabela 11. Relação de <i>features</i> por componente definida pelo arquiteto do projeto.....	94
Tabela 12. Relação de <i>features</i> por componente corrigida.	98
Tabela 13. Relação de <i>features</i> por componente do Ginga-CC.	102
Tabela 14. Resultados da execução do algoritmo.....	107

Lista de Siglas

ADL	<i>Architecture Description Language</i>
CMM	<i>Component-Module Matrix</i>
COTS	<i>Commercial Off-The-Shelf systems</i>
DBC	Desenvolvimento Baseado em Componentes
DDS	Desenvolvimento Distribuído de Software
DSM	<i>Design Structure Matrix</i>
ED	Engenharia de Domínio
EA	Engenharia de Aplicação
LPS	Linhas de Produto de Software
MBM	<i>Module Based Matrix</i>
SA	<i>Simulated Annealing</i>
SBTVD	Sistema Brasileiro de TV Digital Terrestre
UML	<i>Unified Modeling Language</i>

Capítulo 1

Introdução

Projetos de software variam em escala e complexidade, e envolvem recursos físicos, tecnológicos e humanos, o que significa que desenvolver software não é barato e exige grande esforço de gerenciamento. A disciplina de engenharia de software compreende todos os aspectos da produção de software e tem por objetivo fundamental o desenvolvimento de sistemas de software de alta qualidade dentro de prazos e custos adequados. Nessa disciplina, uma área particularmente interessante é a engenharia de software baseada em reuso, cujo foco está no reuso de software como estratégia para promover a redução de custos e de riscos, bem como do tempo de desenvolvimento.

Nos últimos anos, diversas técnicas foram desenvolvidas para apoiar o reuso de software. Elas exploram o fato de que sistemas em um mesmo domínio de aplicação são similares e têm potencial de reuso, que o reuso é possível em diferentes níveis (a partir de funções simples até aplicações completas) e que padrões para componentes reusáveis facilitam o reuso [Sommerville 2007].

As principais técnicas de reuso existentes na atualidade são [Sommerville 2007]:

- **Padrões de projeto (*design patterns*):** São descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular. Padrões de projeto promovem a uniformização da estrutura do software e definem um vocabulário comum entre desenvolvedores;
 - **Desenvolvimento baseado em componentes,** que possibilita a composição de novos sistemas a partir da integração de componentes de software, sendo necessário o uso de um repositório de componentes a partir do qual componentes são publicados e recuperados;
 - ***Frameworks de aplicação*,** que promove a reutilização de toda a arquitetura de um domínio (projeto e código) a partir de um conjunto de classes abstratas e concretas que
-

podem ser adaptadas e ampliadas para criar novos sistemas, e que também podem ser compostas pela instanciação de vários padrões de projeto;

- **Sistemas orientados a serviços:** Consistem na interligação de serviços independentes, compartilhados e, possivelmente, remotos. Sistemas orientados a serviços estão baseados nos princípios da computação distribuída e utilizam o paradigma requisição/resposta para estabelecer a comunicação entre os sistemas clientes e os sistemas que implementam os serviços;
- **Linhas de produto:** Consistem em um tipo de aplicação generalizado em uma arquitetura comum que pode ser adaptada de diferentes modos para diferentes clientes;
- **Integração de COTS (*Commercial Off-The-Shelf systems*):** Promove o reuso de sistemas completos, mas sobre os quais não há controle sobre funcionalidade, desempenho e evolução;
- **Bibliotecas de programa:** Lidam com o reuso de elementos de software de granularidade muito fina, tais como classes e funções;
- **Geradores de programa:** Possibilitam a geração automática de sistemas completos ou fragmentos de sistema, o que também envolve a reutilização de padrões e algoritmos, mas que têm aplicação limitada a um pequeno número de domínios de aplicação;
- **Desenvolvimento orientado a aspectos:** Ao promover uma melhor separação entre os requisitos funcionais e não-funcionais de um sistema, aumenta a modularização deste, permitindo que funcionalidades sejam reusadas de forma mais eficiente se comparado ao tradicional desenvolvimento orientado a objetos.

Dentre as técnicas de reuso mencionadas, Linhas de Produto de Software (LPS) tem ganhado bastante espaço na indústria nos últimos anos, por promover o reuso de software de maneira sistemática e previsível. Em geral, processos de desenvolvimento de LPS são constituídos por dois sub-processos: *Engenharia de Domínio* (ED) e *Engenharia de Aplicação* (EA). Durante o sub-processo de ED, é construída uma infra-estrutura reusável que compreende a arquitetura de referência da linha de produtos. No sub-processo de EA, por sua vez, a arquitetura de referência é reusada para derivar uma nova aplicação, o que pode envolver a configuração ou a adaptação de componentes existentes, bem como a introdução de novos componentes.

A adoção de linhas de produto oferece inúmeros benefícios, tais como [Brownsword, Clements 1996]: as organizações tendem a se concentrar em um segmento específico de mercado, o que tende a aumentar a sua rentabilidade; a melhoria da qualidade dos produtos, uma vez que os artefatos na plataforma são revisados e testados em muitos produtos e, conseqüentemente, a chance de detectar falhas e corrigi-las é maior; a integração no desenvolvimento incremental é facilitada; o custo e o tempo de desenvolvimento das aplicações são reduzidos consideravelmente; o teste de software é simplificado, porque os testes das partes de uma aplicação também podem ser reutilizados.

Apesar desses benefícios, LPS também apresenta desafios que precisam ser cuidadosamente considerados. Por exemplo, para uma organização iniciar uma linha de produto geralmente é necessário realizar um alto investimento até que a arquitetura de referência seja efetivamente definida e novas aplicações possam ser geradas, para então ser obtido o retorno do investimento realizado. Além disso, gerar uma arquitetura de referência não é tarefa trivial, pois é necessário conhecer em profundidade o domínio da linha de produto para garantir que a arquitetura projetada atende aos requisitos das aplicações que devem reutilizá-la. Para tanto, é preciso que as organizações invistam em profissionais qualificados, legítimos especialistas no domínio da linha de produto em desenvolvimento, que não são baratos e nem sempre estão disponíveis localmente.

Uma abordagem que possibilita encontrar especialistas do domínio e equipes mais qualificadas para o desenvolvimento de software é o desenvolvimento distribuído de software (DDS) [Kobitzsch et al. 2001]. Além disso, essa forma de trabalho ainda contribui com outros benefícios que complementam ou reforçam os benefícios já oferecidos pela técnica de LPS, tais como redução de custo de desenvolvimento e aumento da qualidade dos produtos. De fato, é possível verificar certa tendência de adoção de DDS no desenvolvimento de projetos de software de grande porte, motivada principalmente pela crescente globalização dos mercados e da produção. A própria técnica de linhas de produto oferece suporte ao desenvolvimento de produtos para mercados globais, tornando a indústria de software mais competitiva e minimizando os custos do desenvolvimento em escala global [Paulish 2003].

Por sua vez, abordagens de DDS também têm limitações relacionadas, principalmente, às dificuldades de comunicação entre as equipes participantes de um mesmo projeto. Isso pode ser verificado em [Carmel 1999] e [Herbsleb, Grinter 1999], onde diversas questões discutidas são causadas por problemas de comunicação, o que demonstra a sua relevância. E

de fato, a comunicação adequada no trabalho cooperativo auxilia no gerenciamento das atividades, permite a identificação de falhas e desvios na execução do planejamento, facilita a solução de conflitos, apóia a tomada de decisão e contribui para a união entre equipes.

Nesse cenário, a arquitetura de software é considerada um importante meio de coordenação de equipes. A razão é que este artefato orienta os desenvolvedores em suas decisões, impondo a necessidade de compatibilidade entre elas, desde que exista um entendimento comum sobre o que é a arquitetura [Herbsleb, Grinter 1999; Ovaska et al. 2003]. Além disso, segundo abordado em [Repenning et al. 2001], a arquitetura de software baseada em componentes, em particular, pode ser vista como um motivador para a utilização da abordagem de DDS, uma vez que componentes contribuem significativamente com a modularização da arquitetura. Isso é evidenciado em [Ghezzi et al. 2003], onde é afirmado que quanto mais independente são os componentes de software, mais independentemente pode ser conduzido o trabalho das diversas equipes.

Nesse sentido, geralmente é admitido que se cada componente é desenvolvido por uma única equipe, existe relativamente pouca necessidade de coordenação entre equipes [Herbsleb 2007]. Dessa forma, em muitos projetos componentes são desenvolvidos por equipes remotas como se fossem unidades totalmente independentes, tal como realizado no projeto distribuído de linha de produto no domínio de aplicações para celular descrito em [Paulish 2003]. Entretanto, mesmo em arquiteturas bem modularizadas, componentes sempre mantêm algum nível de dependência entre si, porque componentes devem ser integrados para constituir aplicações.

Por exemplo, em [Sosa, Eppinger et al. 2002] é apresentado um estudo de caso na indústria de telecomunicação em que são analisados os efeitos sobre a comunicação causados pela distância geográfica entre as equipes, a interdependência entre elas, e em particular, como esses fatores influenciam na escolha dos meios de comunicação adotados em uma organização. O estudo levou à constatação de uma forte correlação entre entidades de software dependentes em um sistema de software e a frequência de comunicação entre membros de equipes que lidam com esses componentes.

Estudos etnográficos que também sugerem que as dependências técnicas entre os “pedaços de código” criam “dependência social” entre desenvolvedores de software podem ser verificados em [Staudenmayer 1997; Allen 1997; Grinter 2003; De Souza et al. 2004]. Além disso, conforme discutido em [Grinter et al. 1999], essa situação também pode levar ao

problema da *distribuição de features* ou seja, à separação de componentes que implementam uma mesma *feature*, o que indiretamente significa que componentes dependentes entre si são desenvolvidos por equipes distintas, tornando o projeto vulnerável à problemas de comunicação.

1.1. Motivação

O uso da arquitetura de software como ferramenta de coordenação e veículo de comunicação no desenvolvimento de software não é recente, como pode ser verificado em [Conway 1968; Parnas 1972]. Mas, as particularidades de sua utilização no contexto de desenvolvimento distribuído e, em especial, de linhas de produto, ainda não foram vastamente exploradas, conforme descrito a seguir.

Soluções que visam tratar os problemas de comunicação em DDS geralmente consistem em ferramentas que combinam recursos de áudio, vídeo e comunicação textual para melhorar a qualidade da comunicação entre as pessoas, e quase sempre estão voltadas para o desenvolvimento colaborativo. Esse é o caso da ferramenta RaisAware [Costa et al. 2008], que explora o relacionamento sócio-técnico entre a arquitetura de software e a coordenação do trabalho de desenvolvimento de software através da análise de dependências entre os artefatos produzidos na etapa de codificação e entre as atividades dos desenvolvedores. O desenvolvimento colaborativo consiste em uma área particular do DDS, em que é discutida a questão de pessoas remotas desempenharem uma mesma tarefa de maneira conjunta, compartilhando recursos, como o código fonte de uma aplicação.

Outras soluções no contexto de DDS, tais como técnicas e metodologias de desenvolvimento, embora admitam que a arquitetura de software baseada em componentes possibilita uma melhor separação das atividades a serem desempenhadas por diferentes equipes, não oferecem suporte sobre como efetivamente proceder para tirar vantagem disso. Dessa forma, tais abordagens geralmente ignoram que os componentes interagem entre si e que precisam ser integrados, o que exige intensa comunicação entre as equipes responsáveis por implementá-los, como se uma arquitetura bem modularizada simplesmente fosse o suficiente para garantir comunicação eficiente entre as equipes de desenvolvimento.

Em se tratando do caso específico de LPS que emprega o desenvolvimento distribuído, observa-se que não há muitos estudos que discutem esse tema. Na verdade, a maioria dos processos de LPS não trata das dificuldades de lidar com DDS, e as abordagens que mais se aproximam dessa questão estão voltadas para os aspectos organizacionais de uma linha de

produto, tal como pode ser verificado em [Bosch 2001] e [Linden et al. 2007]. Em geral, tais abordagens tentam categorizar diferentes cenários de desenvolvimento na perspectiva de uma empresa de desenvolvimento de software, destacando a viabilidade de empregá-los e realçando suas vantagens e desvantagens, sem explorar soluções para as desvantagens identificadas.

O presente trabalho visa minimizar essa lacuna e assim, efetivamente contribuir com a minimização dos problemas de comunicação enfrentados pelas equipes de desenvolvimento envolvidas em projetos de LPS, em particular na situação dessas equipes estarem geograficamente dispersas, segundo uma abordagem centrada na arquitetura.

De acordo com [Mockus, Weiss 2001], para que o desenvolvimento de software seja mais eficiente, a distribuição geográfica das equipes e a estrutura de comunicação adotada devem corresponder à divisão de tarefas e, além disso, tarefas fortemente relacionadas, que requerem comunicação e coordenação frequentes, devem ser desenvolvidas no mesmo local. Tal conclusão induz à formulação da hipótese:

Dado que componentes para serem implementados consistem em tarefa de desenvolvimento, a fim de reduzir as necessidade de comunicação e coordenação entre equipes de desenvolvimento, é necessário agrupar componentes fortemente dependentes em módulos fracamente dependentes, alocando cada módulo à uma única equipe.

Em apoio a essa hipótese, a presente dissertação descreve uma abordagem para recomendar módulos fracamente dependentes a partir do agrupamento de componentes de software fortemente dependentes no contexto de projetos distribuídos de LPS. A abordagem compõe o *framework* de recomendações para alocação de equipes de desenvolvimento distribuído de software em projetos de LPS apresentado em [Pereira et al. 2010]. Neste *framework*, as recomendações são geradas com base em três tipos de informação: os módulos de software que devem ser implementados por cada equipe e seus requisitos técnicos; os atributos técnicos das equipes, como experiência de trabalho, conhecimento em tecnologias e títulos; e os atributos não técnicos das equipes, como a língua nativa, fuso-horário e princípios sócio-culturais.

1.2. Objetivos

Partindo da premissa de que dependências entre componentes de software exercem grande influência sobre a necessidade de comunicação entre suas respectivas equipes de desenvolvimento, este trabalho tem por objetivo apresentar uma abordagem para identificar candidatos à módulos para serem desenvolvidos de forma (parcialmente) independente por equipes geograficamente dispersas. Para tanto, a abordagem define os seguintes objetivos específicos: definição de medidas quantitativas que descrevem a dependência entre componentes de software de uma LPS, em nível arquitetural; construção de um algoritmo baseado em metaheurísticas para agrupar componentes em módulos, tratando o agrupamento como um problema de otimização; e definição de uma medida quantitativa que descreve as dependências entre módulos, que deve ser empregada para guiar a alocação das equipes de desenvolvimento aos módulos.

A idéia subjacente é que um módulo deve ser definido de maneira a agrupar componentes de software que são dependentes entre si e consequentemente, que exigem uma comunicação mais intensa entre os desenvolvedores que os implementam. Num contexto de desenvolvimento distribuído, isso sugere que os módulos devem ser o menos dependente possível de outros módulos. Nesse sentido, nessa abordagem um módulo representa a unidade mínima de trabalho de implementação, o que significa dizer que um módulo deve ser desenvolvido por uma única equipe, cujos membros estão alocados no mesmo lugar e seguem as mesmas normas de trabalho. Isso previne a ocorrência de problemas de comunicação que podem surgir no caso de ter pessoas de diferentes equipes desenvolvendo um mesmo módulo, em especial no caso das equipes estarem geograficamente distantes, o que por sua vez pode comprometer a qualidade dos componentes construídos.

Apesar disso, é esperado que os módulos mantenham certa dependência entre si, já que em um dado momento eles devem ser integrados. Além disso, de acordo com as dimensões do projeto e da quantidade de equipes disponíveis, é possível que uma mesma equipe seja responsável por implementar mais de um módulo dentro de um mesmo projeto. Assim, a fim de dar suporte à coordenação dos esforços de desenvolvimento de software, é preciso mensurar o grau de dependência entre os módulos de um projeto.

Por exemplo, considere duas equipes desenvolvendo dois módulos que interagem. As equipes precisam coordenar cuidadosamente os seus trabalhos para garantir que os módulos são compatíveis e podem operar de forma apropriada. Se as equipes estão co-localizadas, o

nível de dificuldade que elas podem enfrentar para manter comunicação entre si é provavelmente inferior ao nível de dificuldade que elas podem enfrentar se estão geograficamente distantes, devido às diversas diferenças envolvidas nesse contexto, tais como diferenças culturais, temporais e tecnológicas. Dessa forma, o grau de interação entre os módulos tem impacto direto no grau de comunicação requerido entre as equipes.

1.3. Organização do Trabalho

O trabalho está organizado como descrito a seguir. No capítulo 2 são introduzidos alguns conceitos fundamentais relacionados à abordagem desenvolvida, a fim de facilitar a sua compreensão. O capítulo 3 apresenta os trabalhos correlatos, destacando as semelhanças e diferenças em relação ao presente trabalho e assim, evidenciando suas principais contribuições. O capítulo 4 apresenta o *framework* em que a abordagem está inserida, de maneira a possibilitar uma melhor compreensão sobre o seu uso para o propósito de dar suporte à atividade de alocação de equipes em projetos distribuídos de LPS. O capítulo 5 apresenta a abordagem em si, descrevendo as medidas quantitativas definidas para avaliar a dependência entre componentes e entre módulos, bem como o algoritmo baseado em metaheurísticas para agrupar componentes em módulos. O capítulo 6 descreve os estudos de caso desenvolvidos para avaliar a abordagem em diferentes aspectos e também demonstrar o seu uso, sendo dois estudos sobre linhas de produto hipotéticas e dois sobre projetos reais de LPS. Por fim, no capítulo 7 são apresentadas as considerações finais, evidenciando as contribuições, limitações e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Uma vez que a abordagem desenvolvida se aplica a linhas de produto e está centrada na arquitetura de software, além de fazer uso de metaheurísticas, neste capítulo são apresentados conceitos fundamentais relacionados a este contexto. Dessa forma, inicialmente é feita uma breve introdução ao desenvolvimento de linhas de produto de software, sendo dada ênfase aos aspectos de variabilidade que tornam a arquitetura flexível. Em seguida, são apresentados conceitos básicos relacionados à arquitetura de software, sendo apresentadas algumas metodologias existentes para análise e avaliação da arquitetura de software que podem ser úteis para a utilização desta como ferramenta de coordenação de atividades de desenvolvimento de software. Então, é apresentada a abordagem *Design Structure Matrix* como forma de representar uma arquitetura de software. Por fim, é feita uma breve introdução à Engenharia de Software Baseada em Buscas, que lida com os problemas de engenharia de software como problemas de otimização.

2.1. Linhas de Produto de Software

A técnica de reuso de software denominada *Linhas de Produto de Software* se propõe a sistematizar a produção de famílias de produtos com base nos conceitos de *plataforma* e *customização*. Uma família de produtos de software consiste em um conjunto de produtos aplicáveis a um segmento específico de mercado, que compartilham entre si um conjunto controlado de características e funcionalidades, salvo algumas variações existentes, que os tornam distinguíveis dos demais [Pohl et al. 2005].

A geração dos membros da família se dá a partir de um conjunto compartilhado de recursos também denominado de *plataforma da linha de produto*, que agrega todos os artefatos que são produzidos nas diversas atividades que compõem um processo de desenvolvimento de software, e que podem ser reusados para toda a linha (artefatos comuns e opcionais). A partir da customização da plataforma de reuso é possível derivar produtos com propriedades específicas, que atendem às necessidades do cliente ao qual estão destinados.

A customização da plataforma de reuso é possível a partir da construção de artefatos flexíveis. Para tanto, a maioria dos processos de LPS estão organizados em sub-processos de Engenharia de Domínio (ED) e de Engenharia de Aplicação (EA). A engenharia de domínio pode ser definida como a atividade de coletar, organizar e armazenar as experiências passadas no desenvolvimento de sistemas ou partes de sistemas num domínio particular na forma de artefatos reusáveis, bem como de oferecer meios adequados para reusar tais artefatos (isto é, qualificação, recuperação, disseminação, adaptação, configuração, etc.) quando da construção de novos sistemas [Czarnecki, Eisenecker 2000]. Dessa forma, é durante o sub-processo de ED que os aspectos comuns e variáveis da linha de produto são identificados.

A concepção da engenharia de domínio advém à definição das linhas de produto de software, compondo apenas uma das fases necessárias para a construção desta, e tem por objetivo possibilitar a sistematização do reuso de software. As LPS estenderam esse conceito associando desenvolvimento *para reuso* e desenvolvimento *com reuso*, ampliando sua aplicabilidade na indústria de software. Assim, processos específicos para engenharia de domínio serviram (e ainda servem) de base para processos de LPS.

Já a engenharia de aplicação se dedica ao estudo das melhores técnicas, processos e métodos para a produção de aplicações no contexto do desenvolvimento baseado em reutilização [Griss et al. 1998]. Em outras palavras, é durante o sub-processo de EA que é realizado o desenvolvimento *com reuso*, onde artefatos do domínio são reusados e os aspectos variáveis da linha de produto são explorados. Com um modelo de domínio claro e bem definido é possível desenvolver a especificação de uma aplicação a partir de componentes previamente definidos, não precisando realizá-la a partir do zero a cada vez que uma nova aplicação num dado domínio precisar ser construída [Gomaa et al. 1992].

Em geral, os processos de engenharia de domínio e de engenharia de aplicação estão organizados segundo as fases de *análise*, *projeto* e *implementação*. A Tabela 1 [Miler 2000] a seguir resume as principais fases de um processo genérico de desenvolvimento de LPS, segundo as atividades indicadas.

Tabela 1. Descrição genérica de um processo de LPS.

	Engenharia de Domínio (ED)	Engenharia de Aplicação (EA)
Análise	Definição do escopo do domínio; Descrição dos componentes do domínio; Modelagem do diagrama de integração dos componentes; Especificação dos modelos estruturais do domínio	Definição do escopo da aplicação; Seleção dos componentes do domínio; Adequação dos componentes do domínio; Especificação de novos componentes
Projeto	Definição das arquiteturas suportadas no domínio; Refinamento dos modelos estruturais; Especificação da interface básica entre os componentes	Definição e Instanciação da arquitetura da aplicação; Refinamento dos modelos estruturais e de interação da aplicação; Adaptação da estrutura dos componentes de domínio; Especificação da interface básica dos novos componentes
Implementação	Codificação dos componentes; Teste dos componentes; Empacotamento do código legado	Adaptação dos componentes de domínio; Desenvolvimento de novos componentes; Instanciação da aplicação; Testes da aplicação

Dentre as fases descritas, as fases de análise e projeto durante o sub-processo de ED são de fundamental importância para o desenvolvimento do presente trabalho, pelo fato de juntas resultarem na construção da *arquitetura de referência* (também denominada *arquitetura de domínio*), o principal artefato da plataforma de reuso, que compõe a estrutura fundamental de todos os produtos da linha, e que difere da visão tradicional da arquitetura de software como uma estrutura única e fixa.

Durante a análise do domínio, os aspectos comuns e variáveis entre as aplicações são identificados e representados segundo a definição de *pontos de variação* (local do artefato de software em que uma decisão de projeto pode ser tomada) e *variantes* (alternativas de projeto associadas ao ponto de variação). Na maioria dos processos de desenvolvimento de LPS isso é feito através da modelagem de um *diagrama de features*, sendo uma *feature* uma propriedade de sistema que é relevante para alguma parte interessada, usada para capturar funcionalidades comuns ou variáveis entre sistemas de uma mesma família de produtos [Czarnecki et al. 2005]. *Features* podem ser obrigatórias ou comuns (toda aplicação precisa incorporá-la), opcionais (somente algumas aplicações a incorporam) e alternativas (há um grupo de *features* relacionadas e restrições que definem a quantidade destas que podem coexistir em uma aplicação).

A Figura 1 a seguir ilustra um exemplo de diagrama de *features*. Trata-se do diagrama de *features* de uma linha de produtos de *middleware* para a computação móvel, que foi tomada como estudo de caso no presente trabalho, conforme apresentado no Capítulo 6. Nessa figura, *Mecanismos de Tolerância a Falhas* define um conjunto de *features* obrigatórias, enquanto que *Paradigmas de Comunicação* representa um conjunto de *features* no qual é obrigatório que ao menos uma das *features* exista na aplicação, podendo existir mais de uma *feature* desse grupo.

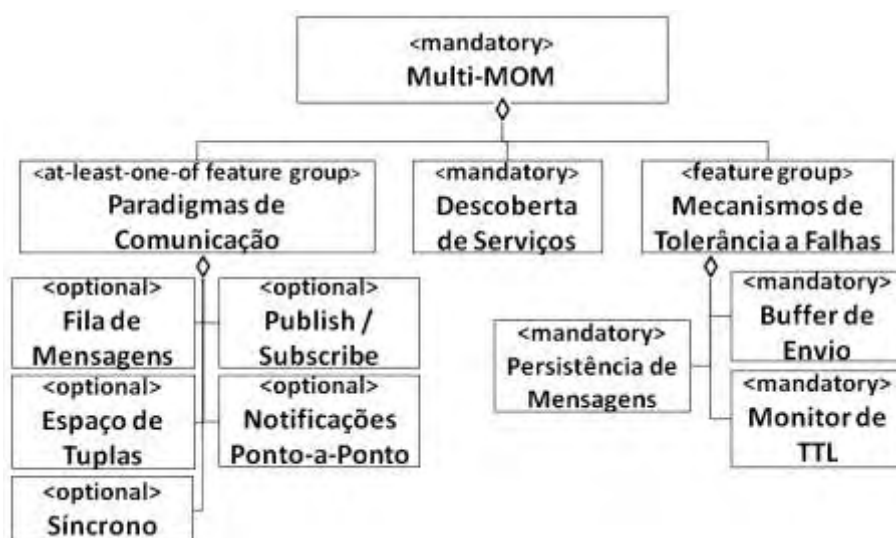


Figura 1. Exemplo de diagrama de *features*.

Durante o projeto do domínio, a arquitetura do domínio é efetivamente construída, o que envolve o mapeamento das *features* para componentes de software, uma vez que o presente trabalho lida com a implementação de LPS segundo a abordagem de DBC, bem como a modelagem da variabilidade em nível arquitetural, abstraindo-se os mecanismos concretos de variabilidade que serão adotados na implementação dos componentes. Como resultado, é realizado o projeto de componentes comuns (presentes em todas as aplicações), componentes opcionais (presentes em algumas aplicações) e componentes variáveis, que tanto podem ser comuns como opcionais.

De acordo com [Linden et al. 2007], existem três técnicas gerais de variabilidade em nível arquitetural: **adaptação**, **substituição** e **extensão**. Na técnica de adaptação existe uma única especificação do componente e uma única implementação do componente, sendo oferecidos meios para ajustar o seu comportamento, tal como um arquivo de configuração ou parametrização em tempo de execução, dentre outros meios. Por exemplo, na linha de produto de microondas descrita em [Gomaa 2004], existe um componente denominado *Keypad*,

responsável pela funcionalidade do teclado de microondas. O teclado de microondas mais básico possui botões numéricos e botões de iniciar (*start*), parar/cancelar (*stop/cancel*) e tempo de cozimento (*cooking time*). Entretanto, microondas mais modernos podem oferecer botões adicionais, como o botão *minute plus*, que quando o microondas está em funcionamento, permite que seja acrescido um minuto ao tempo de cozimento. Na referida linha de produto, o componente *Keypad* foi projetado para ser um componente adaptável, sendo possível habilitar e desabilitar botões através da passagem de parâmetro para o método invocado por esse componente.

Na técnica de substituição, há várias implementações disponíveis para uma mesma especificação de componente, todas elas desenvolvidas com base em uma única especificação genérica que define exatamente o que o componente deve fazer. Cada implementação visa atender às diferentes variantes e, portanto, o número de implementações é limitado pelo número de variantes. No entanto, apesar da especificação genérica em comum, componentes variantes podem incluir interfaces e *features* diferentes, definindo diferentes comportamentos.

Por exemplo, no caso da linha de produto de microondas há um componente composto denominado *Microwave Display*, responsável pela funcionalidade da tela de um microondas e que é constituído por dois componentes variáveis, genericamente denominados *Display Interface* e *Display Prompts*. O componente *Display Interface* recebe comandos para exibir mensagens de aviso, compreendendo dois tipos de componente: *One-line Microwave Display* e *Multi-line Microwave Display*. Ambos os tipos de componente provêm a interface *IDisplay*, que possui quatro operações. Entretanto, uma das operações está habilitada somente para o componente *Multi-Line Microwave Display*, o que significa que apesar desses componentes seguirem a mesma especificação genérica, eles possuem comportamentos diferentes, o que é perceptível não somente pelo fato de terem implementações distintas no geral, mas pelo fato de um prover uma operação que o outro não provê. Já o componente *Display Prompts* é responsável por manter o texto das mensagens de aviso e compreende cinco tipos de componentes, que diferem entre si pelo idioma das mensagens: *English Display Prompt*, *German Display Prompt*, *Spanish Display Prompt*, *Italian Display Prompt* e *French Display Prompt*. Todos esses componentes provêm a interface *IDisplayPrompt*, que possui três operações, estando todas elas habilitadas para todos os tipos de componentes. Nesse caso, os componentes possuem exatamente a mesma especificação, embora possuam implementações diferentes.

Por fim, a técnica de extensão requer que a arquitetura ofereça meios que permitam que novos componentes sejam adicionados, não sendo possível prever a quantidade destes, nem tampouco o tipo do componente (comum ou específico de aplicação), suas interfaces, operações e *features*. Ao longo do ciclo de vida de uma LPS, componentes específicos de aplicação podem ser identificados e até posteriormente introduzidos à arquitetura de domínio, o que constitui um exemplo de extensão. Tais componentes são anexados a arquitetura reusada semelhantemente à *plug-ins* que adicionam funcionalidade a uma aplicação hospedeira.

No caso particular de DBC, é possível ainda prover variabilidade por meio da composição de componentes, o que implicitamente envolve as técnicas gerais de variabilidade descritas. Um componente pode ser composto por diferentes estruturas e combinações de elementos arquiteturais, fornecendo pontos de variação. Por exemplo, no caso da linha de produto de microondas, o componente composto *Microwave Display* anteriormente detalhado é constituído por componentes do tipo *Display Interface* e do tipo *Display Prompts*, os quais são componentes variáveis, que possuem duas e cinco implementações distintas respectivamente. Dessa forma, é possível definir dez implementações distintas para o componente composto *Microwave Display*.

2.2. Arquitetura de Software

A arquitetura de software pode ser entendida como o conjunto de componentes, subsistemas, relacionamentos, interações, propriedades desses elementos e o conjunto de princípios que juntos, constituem as propriedades e restrições fundamentais de um sistema ou conjunto de sistemas de software [IEEE 2000]. Nesse sentido, uma descrição arquitetural consiste em um conjunto de artefatos que documentam a arquitetura de software.

Na abordagem desenvolvida, é dada ênfase à capacidade da avaliação da arquitetura de software de gerar conhecimento útil para a alocação de equipes geograficamente distribuídas às unidades de implementação, em particular no caso de linhas de produto de software, a fim de minimizar os problemas de comunicação comumente identificados nesse cenário. Dessa forma, se faz necessário tomar como entrada uma descrição da estrutura da arquitetura em termos de componentes e seus relacionamentos, a fim de avaliá-los.

No contexto tradicional de software, a estrutura de uma arquitetura pode ser descrita em UML, em termos de seus aspectos estáticos (através de diagramas de classe, de componentes, de subsistemas ou de pacotes) e dos seus aspectos dinâmicos (através de

diagramas de interação, de transição de estados ou de atividade). No caso de LPS, os artefatos empregados são os mesmos, com o diferencial de que uma descrição arquitetural tanto pode representar uma arquitetura de domínio quanto uma arquitetura específica de aplicação, e para tanto são introduzidas as propriedades de variabilidade e opcionalidade mencionadas.

Por sua vez, a avaliação da arquitetura de software visa determinar as propriedades e os pontos fortes e fracos de uma arquitetura de software, de um estilo arquitetural ou de um padrão de projeto. Através da avaliação da arquitetura de software é possível melhorar a compreensão e a documentação do sistema, bem como aprimorar a detecção de problemas e aumentar o aprendizado organizacional [Roy, Graham 2008].

Existem diversas abordagens para a avaliação de uma arquitetura de LPS, que conforme descrito em [Oliveira Junior et al. 2007], podem ser organizadas em três grandes grupos, segundo os objetivos subjacentes:

- **Avaliação de atributos de qualidade:** estudos que contemplam a avaliação de atributos como desempenho, manutenibilidade, reusabilidade, confiabilidade, dentre outros;
- **Avaliação estrutural:** estudos que contemplam a avaliação de arquiteturas de acordo com a sua estrutura em termos de métricas para componentes, interfaces, assinaturas de métodos, utilização de serviços, entre outras;
- **Definição e avaliação de escopo:** estudos que contemplam a análise de riscos e benefícios, adoção e melhoria do processo de LPS, modelos econômicos e de estimativas, entre outros.

Independentemente da categoria na qual estão situadas, as abordagens de avaliação existentes ainda podem ser estudadas de acordo com o tipo de técnica que empregam (qualitativa ou quantitativa) [Etxeberria, Sagardui 2005]. A avaliação qualitativa da arquitetura de software visa responder questões imprecisas, como por exemplo, se o produto atende às necessidades do cliente, ou seja, a arquitetura faz o mapeamento adequado dos requisitos funcionais e não-funcionais para um conjunto de código de implementação. Esse tipo de avaliação geralmente é feito através do estudo de cenários e pela aplicação de questionários e *checklists*. Já a avaliação quantitativa pode ser usada para medir a aplicação de técnicas que englobam qualidades específicas, normalmente operacionais, como por exemplo, o nível de acoplamento entre dois componentes de software, e inclui simulação, prototipagem, modelos matemáticos e experimentais.

As técnicas quantitativas complementam as técnicas qualitativas: a medição de software se dedica a derivar um valor numérico para algum atributo de um produto de software para que, a partir da comparação entre valores de diferentes produtos, possa ser possível inferir sobre a qualidade de software de maneira mais precisa [Sommerville 2007].

No presente trabalho são exploradas as métricas em nível arquitetural com foco na avaliação da dependência entre componentes de software. Na literatura específica de LPS, não foram identificadas métricas para essa finalidade e por isso, foram estudadas métricas arquiteturais em geral (dentro e fora do contexto de LPS), relevantes para a compreensão do relacionamento entre componentes. As principais métricas identificadas são apresentadas a seguir: as métricas de LPS definidas em [Hoek et al. 2003], as métricas arquiteturais baseadas em *responsabilidades*¹ definidas em [Sant'Anna et al. 2007] e as métricas de componentes propostas em [Gill, Balkishan 2008].

Em [Hoek et al. 2003] são apresentadas duas métricas para avaliar a qualidade estrutural de uma arquitetura de LPS com base no conceito de utilização dos serviços expressos em uma ADL (*Architecture Description Language*). Nessa abordagem, o termo serviço designa qualquer tipo de recurso publicamente acessível, tal como métodos ou funções públicas e estruturas de dados diretamente acessíveis. As duas métricas são detalhadas a seguir.

- **PSU (*Provided Service Utilization*)**: Relaciona o número total de serviços oferecidos por um componente e que são realmente utilizados por outros componentes na arquitetura com o número total de serviços oferecidos pelo componente.

$$PSU_x = \frac{P_{\text{real}}}{P_{\text{total}}}$$

- **RSU (*Required Service Utilization*)**: Relaciona o número total de serviços requeridos por um componente e que são realmente oferecidos por outros componentes na arquitetura com o número total de serviços requeridos pelo componente.

$$RSU_x = \frac{R_{\text{real}}}{R_{\text{total}}}$$

Ambas as métricas resultam em valores no intervalo [0,1]. *PSU* define a taxa de utilização dos serviços oferecidos por um componente, enquanto que *RSU* desempenha um

¹O princípio da separação de responsabilidades (*concerns*) foi introduzido em [Dijkstra, 1976], tendo como objetivo dividir o domínio de um sistema em partes menores com o intuito de entender melhor cada parte isoladamente. Uma responsabilidade (*concern*) é alguma parte do domínio do sistema que se deseja tratar como uma unidade conceitual única. No desenvolvimento de software, uma responsabilidade pode ser vista como um requisito funcional ou não funcional de um sistema.

papel complementar, definindo uma taxa de satisfação. Valores de *PSU* menores que 1 (um) indicam componentes que podem estar sendo reusados, e que por isso, oferecem interfaces que não são requeridas por outros componentes em uma dada arquitetura. Componentes obrigatórios, por exemplo, podem oferecer interfaces que são requeridas apenas por componentes opcionais. No entanto, valores de *PSU* muito próximos de 0 (zero) podem significar falta de coesão em um componente, ou seja, o componente agrega mais funcionalidades do que realmente é necessário. No caso de *RSU*, um valor diferente de 1 (um) indica que o componente pode não funcionar apropriadamente em uma dada arquitetura, pois ele requisita interfaces que não são oferecidas por componente algum na arquitetura e, portanto, é desejável que esse valor seja sempre 1 (um). Dessa forma, valores absolutos de *PSU* e *RSU* denotam potenciais problemas em uma dada arquitetura, enquanto que valores relativos (extraídos de duas arquiteturas diferentes, porém relacionadas) servem para examinar e avaliar soluções estruturais alternativas.

As métricas arquiteturais definidas em [Sant'Anna et al. 2007] tratam da questão da difusão de conceitos entre componentes de software, mostrando que esse fenômeno contribui para o aumento da complexidade dos componentes e compromete a coesão destes, o que conseqüentemente causa a redução da modularidade da arquitetura como um todo. As métricas são utilizadas para comparar arquiteturas alternativas para um sistema, podendo ser aplicadas à arquiteturas orientadas a aspectos (AOP) e arquiteturas baseadas em componentes. A Tabela 2 resume as métricas referenciadas, apresentando apenas aquelas que são interessantes para os propósitos do presente trabalho.

Tabela 2. Métricas de modularidade baseada em conceitos.

Atributo	Métrica	Definição
Acoplamento entre componentes	<i>AC (Afferent Coupling Between Components)</i>	Conta o número de componentes que requerem serviço do componente avaliado.
	<i>EC (Efferent Coupling Between Components)</i>	Conta o número de componentes dos quais o componente avaliado requer serviços.
Coesão de componente	<i>LCC (Lack of Concern-based Cohesion)</i>	Conta o número de conceitos tratados pelo componente avaliado.
Complexidade de interface	Número de Interfaces	Conta o número de interfaces de cada componente.
	Número de Operações	Conta o número de operações nas interfaces de cada componente.

As métricas de acoplamento entre componentes caracterizam duas formas de se avaliar o acoplamento: na perspectiva do quanto um componente requer dos demais e do quanto um componente oferece para os demais, semelhante às métricas *PSU* e *RSU*. A métrica de coesão de componente sugere que quanto maior o número de conceitos tratados por um componente, menor é a sua coesão e, portanto, pode ser interpretada como uma medida de falta de coesão. As métricas de complexidade de interface visam expressar a complexidade de desenvolver um componente segundo o número de interfaces e de operações por interface que este oferece. A lógica é que quanto maior forem esses valores para um dado componente, mais complexo este seja considerado.

Finalmente, em [Gill, Balkishan 2008] são definidas as métricas (complementares) denominadas *Component Dependency Metric (CDM)* e *Component Interaction Dependency Metric (CIDM)*, cujo objetivo final é identificar o impacto da dependência entre componentes sobre a complexidade de software. Essas métricas são empregadas com base em grafos direcionados e matrizes de adjacências. Um grafo direcionado pode ser definido como $G = (N, E)$, onde N é um conjunto finito de vértices (chamados componentes) e E é o conjunto de dependências (caminhos) entre componentes. Uma matriz de adjacências $AM_{n \times n}$ pode representar o grafo G definindo o valor 1 para cada elemento $AM[i, j]$ onde o componente C_i está diretamente conectado ao componente C_j e valor 0, caso contrário.

A partir de um grafo desse tipo, o $path(i, j)$ de um componente C_i em relação a um componente C_j pode ser computado como sendo o número de todos os caminhos no grafo que levam de C_i a C_j . D_i representa a dependência de um componente C_i como sendo a soma de todos os valores de $path$ do componente i em relação aos demais componentes do grafo. A métrica *CDM* pode ser computada como a soma dos valores de D_i de cada componente i do grafo. No pior caso, o valor de *CDM* pode ser computado como:

$$CDM_{max} = n(n-1) \sum_{i=0}^{n-2} \frac{(n-2)!}{(n-2-i)!}$$

Com base nisso, a métrica *Dependency Oriented Complexity Metric (DOCM)* de um sistema pode ser definida como a razão entre *CDM* e CDM_{Max} . O valor de *DOCM* está no intervalo $[0, 1]$, e quanto mais próximo de 0 está esse valor, significa que menor é o número de interações entre os componentes, e portanto, menor é a complexidade e o acoplamento da arquitetura resultante. A métrica *CIDM* é a razão entre o número total de interações diretas entre componentes e o número total de componentes. Valores elevados de *CIDM* indicam uma arquitetura de sistema complexa, com componentes bastantes dependentes entre si.

As métricas apresentadas não podem ser aplicadas para os propósitos da abordagem desenvolvida, embora tenham inspirado o desenvolvimento das métricas discutidas no Capítulo 5. As métricas *PSU* e *RSU* avaliam um componente em relação à arquitetura como um todo, enquanto que as métricas de coesão e complexidade avaliam um componente individualmente, e as métricas *CDM* e *CIDM* avaliam a arquitetura como um todo.

No presente trabalho existe o interesse de se avaliar o relacionamento entre pares de componentes, tal como sugerido pelas métricas *AC* e *EC*. Entretanto, tais métricas consistem na contagem absoluta dos serviços providos e requeridos entre componentes, o que seria mais adequado ser feito numa perspectiva relativa a cada componente e, além disso, não é estabelecido um claro relacionamento entre essas métricas. Por exemplo, se um componente *A* requer 1 interface do componente *B* e um terceiro componente *C* requer 2 interfaces do componente *B*, considerando apenas o valor absoluto de interfaces requeridas de *A* e *C* em relação a *B*, seria possível afirmar que a dependência de *C* para *B* é maior do que a dependência de *A* para *B*. Entretanto, relacionando o total de interfaces requeridas pelos componentes *A* e *C* (incluindo interfaces providas por outros componentes que não *B*), temos que *A* requer o total de 2 interfaces, sendo 1 delas provida por *B*, e que *C* requer o total de 3 interfaces, sendo 2 delas providas por *B*. Então, com base em tais informações, seria possível dizer que a dependência de *A* para *B* é maior do que a dependência de *C* para *B*, pois do total das interfaces requeridas pelos componentes *A* e *C*, *B* tem contribuição mais significativa em relação à *A* do que à *C*.

2.3. *Design Structure Matrix*

Uma técnica que tem sido explorada nos últimos anos no contexto de desenvolvimento de software para avaliação da modularidade da arquitetura de sistemas é a construção e aplicação de métricas à *Design Structure Matrix* (DSM).

A DSM foi originalmente formulada para dar suporte à melhoria de processos de desenvolvimento de produtos no contexto da engenharia mecânica, como a indústria automobilística. Nesse contexto, através de uma DSM é possível representar a estrutura de um projeto através de uma matriz quadrada e analisar as dependências entre as tarefas (ou parâmetros) a serem executadas [Albin 2003]. As tarefas consideradas são de natureza diversa: planejamento, execução, manutenção, dentre outras. A idéia é definir agrupamentos de tarefas de acordo com o nível de dependência entre elas, a fim de facilitar o fluxo de informações dentro de um projeto de desenvolvimento de produto.

Existem dois tipos de DSM, que diferem entre si pela maneira de representação da dependência: DSM *binária*, que expressa a dependência através de um símbolo ou valor binário (tal como “0” e “1”), e DSM *numérica*, que expressa a dependência através de um valor numérico que consiste no peso da dependência.

A Figura 2 consiste em um exemplo hipotético de DSM numérica. De acordo com a Figura 2a, é possível identificar uma relação de interdependência entre os elementos E_1 e E_3 , e entre os elementos E_0 e E_2 . A DSM reordenada da Figura 2b enfatiza tais relacionamentos.

	E_0	E_1	E_2	E_3
E_0	0	0	1	0
E_1	0	0	0	2
E_2	3	0	0	0
E_3	0	1	0	0

(a) DSM inicial.

	E_0	E_2	E_1	E_3
E_0	0	1	0	0
E_2	3	0	0	0
E_1	0	0	0	2
E_3	0	0	1	0

(b) DSM reordenada.

Figura 2. Exemplo de DSM.

Apesar de a DSM numérica representar de maneira mais precisa a dependência, em muitos projetos a DSM binária é preferida, porque é mais fácil de ser construída e porque o cálculo da dependência em si não é padronizado, podendo variar bastante de acordo com o tipo do produto, e dos interesses envolvidos na avaliação da dependência. Dessa forma, o cálculo geralmente é deixado a cargo de um profissional (ou equipe de profissionais) que tem maior experiência no desenvolvimento do produto em questão, sendo definida uma escala de dependência (por exemplo: “1 – fraco”, “2 – moderado”, “3 – forte”), resultando em uma avaliação qualitativa, totalmente subjetiva. Em se tratando de DSM numérica, ainda é possível que esta seja multidimensional, ou seja, cada uma de suas células pode conter mais de um valor numérico, a fim de representar diferentes tipos de dependência, conforme ilustrado em [Helmer et al. 2008].

A utilização de DSM no contexto de desenvolvimento de produtos de software tem sido investigada por grupos de pesquisa que em geral, consideram essa técnica bastante valiosa para caracterizar a arquitetura de software. Nesse caso, ao invés de tarefas, as linhas e colunas na matriz podem representar elementos arquiteturais, como componentes ou aspectos, sendo o relacionamento entre eles definido apenas como existente ou não, através de uma DSM binária, ou por valores numéricos derivados de expressões matemáticas, através de uma DSM numérica.

Por exemplo, em [Sullivan et al. 2001] e em [Lopes, Bajracharya 2005] diferentes alternativas de arquitetura para um mesmo sistema são representadas em DSM como estratégia para facilitar a comparação e posterior tomada de decisão sobre qual alternativa adotar. No primeiro caso, a DSM é preenchida com valores que denotam a dificuldade de se substituir um componente dentro da arquitetura, cujo cálculo é feito com base no relacionamento entre componentes e na análise de projetos nos quais ele foi utilizado, o que só pode ser feito após o componente ser implementado. No segundo, as arquiteturas avaliadas estão baseadas em aspectos e não em componentes.

Estudo similar pode ser visto em [MacCormack et al. 2008], onde representações arquiteturais em DSM são comparadas a fim de serem identificadas diferenças em termos de modularização entre projetos *open source* e projetos de propriedade privada. Nesse estudo, embora sejam considerados componentes de software, o relacionamento entre eles é expresso por valores numéricos calculados com base no código de implementação dos componentes.

Outro exemplo de utilização de DSM no desenvolvimento de produtos de software pode ser verificado em [Sangal et. al 2005], onde é descrita uma abordagem para minimizar os desvios existentes entre projetos de software e suas implementações ao longo do seu ciclo de vida. Nesse exemplo, os elementos representados na DSM do projeto consistem em classes de implementação na linguagem Java e as dependências consistem em referências explícitas de uma classe aos elementos internos à outra classe, identificadas com auxílio de uma ferramenta para inspeção do código fonte das classes.

O presente trabalho utiliza DSM numérica para representar as relações de dependência entre componentes de software em nível arquitetural, em uma etapa que precede sua implementação. Para tanto, oferece métricas para construí-la segundo uma avaliação quantitativa e objetiva, minimizando as dificuldades para a sua utilização. A justificativa consiste na necessidade de identificar níveis de dependência entre módulos para guiar a alocação de equipes de desenvolvimento aos módulos no *framework* no qual está inserida a presente abordagem.

2.4. Engenharia de Software Baseada em Buscas

A Engenharia de Software Baseada em Buscas, tal como seu nome sugere, lida com problemas da engenharia de software como problemas de busca, e procura utilizar algoritmos de otimização para automatizar a construção de soluções para tais problemas [Harman, Jones

2001]. Trata-se de uma vertente de pesquisa relativamente nova dentro da Engenharia de Software, que gradativamente tem conquistado relevância por sua vasta aplicabilidade.

A Tabela 3 enumera os principais problemas de busca nas diferentes áreas da engenharia de software aos quais se aplicam algoritmos de otimização, com base na revisão bibliográfica documentada em [Harman, Jones 2001]. Conforme pode ser observado, os problemas identificados estão espalhados por todo o ciclo de vida dos sistemas de software.

No planejamento de projeto, existe o problema de alocação de recursos decorrente das restrições das tarefas de projeto que devem ser realizadas, tais como prazo, orçamento e prioridade, das interdependências entre as tarefas, e da disponibilidade de recursos. Esse problema é genericamente denominado de *problema de construção de cronogramas de projetos*, podendo receber denominações específicas conforme o tipo de tarefas consideradas, como tarefas de um processo de desenvolvimento (análise, projeto, implementação, testes e manutenção). Outros problemas discutidos consistem na estimativa de tamanho de um projeto dados os seus requisitos, por exemplo, e na estimativa de custo de um projeto, dado o tamanho deste, o esforço envolvido, seus atributos de qualidade e o cronograma planejado.

Tabela 3. Problemas de busca em engenharia de software.

Área	Problemas
Planejamento de projeto	Alocação de recursos, Estimativa de tamanho, Estimativa de custo
Engenharia de requisitos	Seleção de requisitos, Priorização de requisitos
Projeto da arquitetura	Melhoria de qualidade da arquitetura
Implementação	Otimização de código-fonte
Testes	Seleção de casos de teste, Priorização de casos de teste, Geração de dados de teste

Já na engenharia de requisitos, um dos principais problemas de busca investigados consiste na seleção do conjunto de requisitos que deve estar presente para a próxima iteração de desenvolvimento, o denominado *problema da próxima versão*, no qual devem ser balanceadas as necessidades do cliente, as restrições de recursos e as interdependências entre requisitos. Diretamente relacionado a este problema, existe a questão de ordenação de requisitos de acordo com a relevância ou prioridade destes como parte da atividade de planejamento de uma iteração de desenvolvimento.

Em se tratando de projeto arquitetural, um problema recorrente é a melhoria de um projeto arquitetural segundo a abordagem de engenharia reversa, através da qual os limites

dos elementos arquiteturais são redefinidos a fim de melhorar atributos de coesão e acoplamento da arquitetura como um todo.

Na atividade de implementação, embora ainda seja restrita a pesquisa no contexto de engenharia de software baseada em buscas, verifica-se a aplicação de algoritmos de busca em compiladores a fim de encontrar sequências de otimização que resultem em códigos de tamanho reduzido.

Por fim, na área de testes, na qual foram publicados os primeiros trabalhos no contexto de engenharia de software baseada em buscas, os principais problemas dizem respeito à seleção e priorização de casos de teste, semelhantemente ao que acontece na engenharia de requisitos, e na geração de casos de teste, que consiste na identificação de dados de entrada válidos para uma aplicação de acordo com os critérios de teste.

Em algumas destes problemas são utilizadas técnicas tradicionais de busca, como busca exaustiva e *branch-and-bound*. No entanto, devido à complexidade dos problemas, nem sempre é possível aplicar as técnicas tradicionais para encontrar uma solução ótima, devido ao elevado custo computacional envolvido. Nesse sentido, a aplicação de *metaheurísticas* tem se apresentado frequentemente como solução.

Uma metaheurística é um conjunto de conceitos que são usados para definir métodos heurísticos que podem ser aplicados a uma grande variedade de problemas. Na engenharia de software baseada em buscas, as metaheurísticas mais comumente adotadas são *Hill Climbing*, *Simulated Annealing* (SA) e Algoritmos Genéticos [Harman, Jones 2001]. Por padrão, a metaheurística *Hill Climbing* parte de uma solução inicial e a cada iteração gera uma solução na vizinhança da solução atual. Se a nova solução melhorar o valor da função objetivo, a nova solução é aceita, o que significa que a busca procede a partir dela. Caso contrário, a solução é rejeitada e é gerada uma nova solução a partir da solução corrente. A busca prossegue até que nenhuma melhora significativa sobre o valor da função objetivo seja alcançada, um número fixo de iterações seja atingido, ou alguma outra condição de parada seja satisfeita.

A metaheurística *Simulated Annealing* (SA) [Kirkpatrick et al. 1983], empregada no presente trabalho, simula o método de recozimento usado na metalurgia, onde um metal é aquecido até o seu ponto de fusão e em seguida é resfriado lentamente até solidificar-se novamente, atingindo o seu estado de energia mínima. Quando aplicado a problemas de otimização, SA permite que o algoritmo de busca aceite soluções ruins, a fim de melhor explorar o espaço de busca. A probabilidade de aceitação é gradativamente reduzida ao longo

da busca, de maneira que quanto mais lenta for essa redução, maior é a possibilidade do algoritmo resultar na solução ótima ou em uma solução próxima desta.

Por fim, algoritmos genéticos procuram imitar o processo de evolução observado na natureza, com o intuito de encontrar soluções próximas da ótima para problemas de elevada complexidade. Dessa forma, essa estratégia consiste de uma representação das soluções sob a forma de *genes*, um conjunto de operadores genéticos para gerar novos indivíduos (soluções) e uma função de avaliação para determinar quais os indivíduos mais capazes, ou seja, as soluções mais próximas da ótima [Davis 1991].

O presente trabalho define um algoritmo que emprega a metaheurística SA para agrupar componentes de software em módulos. O objetivo do algoritmo é agrupar componentes segundo o nível de dependência existente entre eles, de maneira que componentes fortemente dependentes sejam agrupados a fim de definir módulos fracamente dependentes entre si, que possam ser desenvolvidos por equipes distintas com reduzida necessidade de comunicação. Ao contrário de algoritmos aplicados ao projeto da arquitetura visando melhorar a sua qualidade simplesmente, o algoritmo desenvolvido utiliza o projeto arquitetural para dar suporte à atividade de alocação de equipes de desenvolvimento, contribuindo com a área de planejamento de projeto (Tabela 3). O Capítulo 5 apresenta em detalhes o algoritmo definido neste trabalho.

2.5. Considerações Finais

Este capítulo apresentou as principais áreas de estudo envolvidas no presente trabalho e cujo conhecimento se faz necessário para o seu entendimento como um todo: *Linhas de Produto de Software*, com ênfase no aspecto de variabilidade, que impõe diferenças em relação ao desenvolvimento tradicional de software quanto à avaliação da dependência entre componentes; *Arquitetura de Software*, com foco no uso de métricas para avaliar a dependência entre componentes e no uso de DSM para representar a arquitetura; e *Engenharia de Software Baseada em Buscas*, com ênfase no uso de metaheurísticas como estratégia de busca. Dado que a literatura é ampla a cerca de tais áreas, o capítulo focou nos tópicos relevantes para o trabalho, mas leituras adicionais são fortemente recomendadas para um estudo aprofundado sobre cada uma das áreas de pesquisa.

Capítulo 3

Trabalhos Relacionados

Neste capítulo são apresentadas abordagens identificadas na literatura que estão relacionadas ao presente trabalho. Apesar dessas abordagens não terem o propósito final de dar suporte à alocação de equipes em projetos distribuídos de linhas de produto, elas lidam com o agrupamento de componentes baseando-se na dependência existente entre eles. São elas: uma abordagem inserida em um processo de engenharia de domínio; uma abordagem para agrupamento de itens de trabalho no contexto de DDS; abordagens aplicadas à área de engenharia reversa e que utilizam *Module Dependency Graph* (MDG) para representar dependências entre componentes; e abordagens aplicadas à engenharia mecânica que utilizam DSM para representar dependências entre tarefas, equipes ou componentes físicos, aqui genericamente denominados “elementos”.

3.1. Agrupamento de Componentes na Engenharia de Domínio

A necessidade de agrupar componentes em projetos de LPS é discutida em [Blois et al. 2005] com o objetivo de refinar uma arquitetura de domínio, de acordo com o processo de engenharia de domínio denominado CBD-Arch-DE. Para tanto, essa abordagem apresenta alguns critérios de acoplamento que sugerem o agrupamento de componentes. São eles:

- **Acoplamento por Contextos do Domínio (ACD):** Durante a análise de domínio, *features* são organizadas em diferentes contextos, podendo pertencer a um ou mais contextos. *Features* são utilizadas para derivar tipos de negócio, casos de uso e componentes. De acordo com o critério ACD, para serem melhor compreendidas e reusáveis, as arquiteturas de referência de um domínio deveriam agrupar, tanto quanto possível, componentes que estão relacionados a um mesmo contexto, e evitar agrupamentos de componentes que pertencem a contextos diferentes.
 - **Acoplamento por Componentes de Processo (ACP):** Este critério sugere agrupar componentes de processo com os componentes de negócio dos quais eles requerem interfaces.
-

- **Acoplamento por Interfaces Providas (AIP):** Se um componente fornece muitas interfaces, é sugerido o agrupamento com os componentes que as requerem.
- **Acoplamento por Interfaces Requeridas (AIR):** Se um componente requer muitas interfaces, então é sugerido agrupá-lo com o conjunto de componentes que provêem estas interfaces.
- **Acoplamento por Componentes Requeridos (ACRd):** Se um componente requer dois ou mais componentes, é sugerido o agrupamento dos mesmos. Por exemplo, se um componente A requer duas interfaces do componente B, então o valor $AIR(A)=2$, enquanto que o valor $ACRd(A) = 1$, pois A requer o componente B.
- **Acoplamento por Componentes requerendo outros Componentes (ARnC):** Se um componente é requerido por outros, então o agrupamento destes componentes é sugerido. Por exemplo, se o componente A requer duas interfaces do componente C e o componente B requer uma terceira interface do componente C, então o valor de $ARnC(C) = 2$, pois os componentes A e B requerem o componente C.

Embora os critérios tenham sido propostos num contexto de ED, a abordagem sugere que a maioria deles pode ser usada em qualquer atividade de engenharia de software para ajudar no agrupamento de componentes. Entretanto, aqui é questionada a usabilidade desses critérios na prática, pois algumas características identificadas mostram que sua utilização tende a ser bastante limitada:

- Os critérios ACD e ACP devem ser empregados segundo a descrição arquitetural definida pelo processo CBD-Arch-DE, na qual devem ser identificados componentes de negócio, de processo, utilitário e de infra-estrutura, organizados segundo o estilo arquitetural em camadas. Isso significa dizer que para que esses critérios sejam usados, indiretamente é exigido que o processo de desenvolvimento no qual eles estão baseados seja adotado.
 - Os demais critérios (AIP, AIR, ACRd e ARnC) já são admitidos na literatura de análise de acoplamento entre componentes, conforme pode ser visto em [Hoek et al. 2003], onde são propostas métricas para avaliar a arquitetura de LPS, sendo considerado o acoplamento entre componentes segundo a visão dos serviços oferecidos e requeridos entre eles. Contudo, da forma que foram colocados, esses
-

critérios tendem a gerar agrupamentos que se sobrepõem e não são oferecidas orientações sobre como ponderá-los, tornando a atividade de agrupamentos um tanto subjetiva. Por exemplo, os critérios AIP e AIR tratam da questão de um componente oferecer ou prover “muitas interfaces”. Contudo, o significado de “muito” é subjetivo e, além disso, relativo a uma dada arquitetura. No presente trabalho, portanto, esses critérios são direta ou indiretamente considerados, mas são refinados.

- Os critérios geram elementos arquiteturais de maior granularidade, a partir dos quais é esperado um aumento da qualidade da arquitetura. No entanto, nenhuma avaliação quantitativa foi conduzida no intuito de verificar essa suposição.
- A abordagem determina que, como os componentes avaliados possuem propriedades próprias de variabilidade, opcionalidade e regras de exclusão mútua, após ser adotada alguma sugestão de agrupamento, também devem ser atribuídas ao elemento arquitetural gerado tais propriedades, segundo a combinação ou generalização das propriedades individuais dos componentes. Por exemplo, se o agrupamento sugerido contiver pelo menos um componente obrigatório, então o elemento arquitetural gerado deve ser considerado obrigatório também. Essa medida, contudo, não parece fazer sentido, já que a propriedade de cada componente individualmente não é preservada após o agrupamento, o que pode comprometer o significado da arquitetura.

Em [Blois et al. 2005] ainda é comentado o desenvolvimento de uma ferramenta denominada *GroupCriteria*, que tem por objetivo implementar os critérios de agrupamento descritos, ajudando na visualização dos efeitos causados por eles, e assim auxiliando o engenheiro de software na tomada de decisão sobre quais agrupamentos adotar. Apesar disso, como essa ferramenta foi projetada para ser utilizada no ambiente *Odyssey*, não é possível afirmar que ela pode ser utilizada fora desse ambiente, o que impõe limites à utilização dos critérios citados.

3.2. Agrupamento de Componentes em DDS

Partindo da premissa de que itens de trabalho que requerem comunicação e coordenação frequentes devem ser desenvolvidos em um mesmo lugar, a fim de evitar o comprometimento da qualidade dos produtos, bem como o aumento do tempo e do custo de desenvolvimento, em [Mockus, Weiss 2001] é apresentada uma abordagem para identificar itens de trabalho relacionados entre si no contexto da atividade de manutenção de software.

A abordagem apresenta evidências quantitativas de que requisições de modificação no código fonte de um sistema que é mantido por equipes remotas geram problemas de comunicação e coordenação, dado que o código está espalhado em vários lugares e modificações em uma parte do código podem afetar outras partes relacionadas. Visando reduzir tais problemas, a abordagem oferece um algoritmo para automaticamente identificar partes de código relacionadas denominadas *chunks*, a fim de que estas sejam mantidas em um só local.

O algoritmo está baseado na metaheurística *Simulated Annealing*, tal como o presente trabalho, e iterativamente tenta encontrar uma solução de agrupamento melhor segundo algum critério previamente definido com base no histórico de desenvolvimento, tal como o número de requisições de modificação relacionadas à um *chunk*. O objetivo é encontrar uma solução em que cada requisição de modificação se relaciona com um único *chunk*.

Dado que em [Mockus, Weiss 2001] não é apresentada a função objetivo usada no algoritmo para quantificar os critérios de avaliação, nem qualquer outra informação que permita compreender os detalhes de seu funcionamento, não é possível efetivamente avaliar esta abordagem para compará-la com o presente trabalho.

3.3. Agrupamento de Componentes Aplicado à Engenharia Reversa

Em se tratando da automatização do agrupamento de componentes no desenvolvimento de software, os algoritmos existentes, em sua maioria, são aplicados à área de reengenharia de sistemas. Tais algoritmos geralmente se baseiam em um grafo MDG para representar dependências entre componentes de software, as quais são extraídas do código de implementação. Um grafo MDG consiste em um grafo cujos nós representam os componentes e as arestas (ou conexões) entre estes representam relações de dependência, as quais podem ser ponderadas ou não.

Abstraindo-se a estratégia usada para definir as dependências e representá-las, os algoritmos para agrupamentos de componentes empregados têm forte relação com o presente trabalho, conforme descrito a seguir. A Tabela 4 resume os principais algoritmos identificados para agrupamento de componentes aplicados à engenharia reversa. As subseções 3.3.1 e 3.3.2 a seguir comentam a função objetivo e a estratégia de busca adotada por esses algoritmos.

Tabela 4. Algoritmos para agrupamento aplicados à engenharia reversa.

	Representação de dependência	Função objetivo	Estratégia de busca
[Mancoridis et al., 1998]	Grafo MDG não-ponderado	Função MQ	Busca exaustiva, <i>Hill Climbing</i> , Algoritmos Genéticos
[Doval et al. 1999]	Grafo MDG não-ponderado	Função MQ	Algoritmos Genéticos
[Mancoridis et al., 1999]	Grafo MDG não-ponderado	Função MQ	<i>Hill Climbing</i>
[Harman et al, 2002]	Grafo MDG não-ponderado	Define uma função própria com base em medidas de coesão e acoplamento	Algoritmos Genéticos
[Mitchell e Mancoridis 2002]	Grafo MDG ponderado e não-ponderado	Função MQ atualizada	<i>Hill Climbing</i> , <i>Simulated Annealing</i>
[Mahdavi et al., 2003]	Grafo MDG ponderado e não-ponderado	Função MQ atualizada	<i>Multiple Hill Climbing</i>
[Harman et al., 2005]	Grafo MDG não-ponderado	Função MQ e Função EVM	<i>Hill Climbing</i>
[Praditwong et al., 2010]	Grafo ponderado e não-ponderado	Estratégias MCA e ECA	Algoritmos Genéticos

3.3.1. Função Objetivo

Os algoritmos apresentados em [Mancoridis et al. 1998], [Mancoridis et al. 1999] e [Doval et al. 1999] adotam a função de otimização MQ (*Modularization Quality*) em sua versão original para expressar o valor de uma solução de agrupamento de componentes. O cálculo de MQ para um grafo MDG que contém n agrupamentos consiste na diferença entre a média das conectividades internas (A) e externas (E) de cada agrupamento n , conforme descrito pela Equação (1).

$$MQ = \begin{cases} \frac{\sum_{i=1}^k A_i}{k} - \frac{\sum_{i,j=1}^k E_{i,j}}{\frac{k \times (k-1)}{2}}, & \text{se } k > 1 \\ A_1, & \text{se } k = 1 \end{cases} \quad (1)$$

De acordo com a Equação (2), a conectividade interna A_i de um agrupamento i que contém N_i nós e k_i arestas internas é a razão de k_i sobre o número máximo de arestas internas de i (isto é, N_i^2). Já a conectividade externa $E_{i,j}$ entre dois agrupamentos distintos i e j , que possuem respectivamente N_i e N_j nós, e $\varepsilon_{i,j}$ arestas entre i e j é a razão entre $\varepsilon_{i,j}$ e o número máximo de arestas entre i e j , tal como descrito pela Equação (3).

$$A_i = \frac{\mu_i}{N_i^2} \quad (2)$$

$$E_{i,j} = \begin{cases} 0, i = j \\ \frac{\varepsilon_{i,j}}{2 \times N_i \times N_j}, \text{ caso contrário} \end{cases} \quad (3)$$

São várias as limitações de uso da função MQ: ela admite ser possível a existência de dependência entre um componente com ele próprio; ela está baseada em grafos não-ponderados, o que é equivalente a uma DSM binária, significando que o seu grau de precisão é inferior ao da função definida no presente trabalho; por fim, ela não considera o tamanho dos agrupamentos, aspecto importante para a presente abordagem, uma vez que agrupamentos representam carga de trabalho para suas respectivas equipes de desenvolvimento.

Uma nova versão da função MQ é adotada em [Mitchell, Mancoridis 2002] e [Mahdavi et al. 2003]. Essa nova versão considera grafos ponderados e consiste no somatório do valor da função *Cluster Factor* (CF) de cada agrupamento existente, tal como descrito pela Equação (4). A idéia da nova versão da função MQ é maximizar a conectividade interna dos agrupamentos, a fim de reduzir a conectividade externa entre eles.

$$MQ = \sum_{i=1}^k CF_i \quad (4)$$

O valor de CF_i para um dado agrupamento i está no intervalo $[0,1]$ e seu cálculo está baseado na razão normalizada entre o total do peso das arestas internas a um agrupamento μ_i e a metade do total do peso das arestas externas dos agrupamentos, conforme descrito pela Equação (5). As arestas externas entre dois agrupamentos i e j consiste na soma dos valores de $\varepsilon_{i,j}$ e $\varepsilon_{j,i}$. A equação considera a metade do peso das arestas externas aos agrupamentos visando aplicar uma penalidade igual a ambos os agrupamentos que estão conectados por uma aresta externa.

$$CF_i = \begin{cases} 0, \text{ se } \mu_i = 0 \\ \frac{\mu_i}{\mu_i + \frac{1}{2} \times \left[\sum_{j=1, j \neq i}^k (\varepsilon_{i,j} + \varepsilon_{j,i}) \right]}, \text{ caso contrário} \end{cases} \quad (5)$$

A partir dos experimentos relatados em [Harman et al. 2005] foi identificada uma fragilidade nessa nova versão da função MQ. Se for considerado um grafo MDG perfeito com n agrupamentos, o valor ótimo global de MQ deveria ser n , pois o valor de CF para cada

agrupamento seria 1 (o máximo valor possível, que indica que a conectividade interna de um agrupamento é máxima e a conectividade externa é mínima). Entretanto, o valor de MQ obtido nessa situação foi superior ao esperado, como se existisse $n + 1$ agrupamentos, todos com conectividade interna elevada (próxima à 1) e reduzida conectividade externa.

Outras funções de otimização são propostas em [Harman et al. 2002] e [Harman et al. 2005]. A primeira consiste na média entre os valores das funções de *coesão*, *acoplamento* e *granularidade* de uma solução de agrupamento. A *função de coesão* diz respeito ao peso das arestas internas aos agrupamentos e cujo valor se objetiva maximizar, enquanto que a *função de acoplamento* está relacionada ao peso das arestas entre agrupamentos, cujo valor deve ser minimizado. Já a *função de granularidade* diz respeito ao número de agrupamentos a serem gerados, sendo objetivo da busca encontrar uma solução que satisfaça esse valor ou se aproxime dele o máximo possível. Como a expressão matemática da função não foi representada em [Harman et al. 2002], ela também foi omitida no presente trabalho.

Ao fixar o número de agrupamentos a serem gerados, essa função restringe o espaço de busca do problema, de forma que soluções melhores na perspectiva de maximização do peso das arestas internas e minimização do peso das arestas externas aos agrupamentos podem ser desprezadas. No presente trabalho, o número de agrupamentos a serem gerados é entendido como parte da busca e, portanto, consiste em um valor imprevisível, o que significa que o uso dessa função é inapropriado.

O cálculo da função proposta em [Harman et al. 2005] para uma solução de agrupamento C , denominada *Evaluation Metric* (EVM), consiste na soma da pontuação $h(c_i)$ de cada agrupamento c_i em C , conforme mostrado pela Equação (6).

$$EVM(C) = \sum_{i=1}^m h(c_i) \quad (6)$$

O valor de $h(c_i)$ para um agrupamento c_i é calculado pela Equação (7), na qual k_i consiste no tamanho do agrupamento c_i e $L(c_{xy}, c_{pq})$ consiste na avaliação do relacionamento entre o elemento c_{xy} no agrupamento c_x e o elemento c_{pq} no agrupamento c_p , conforme definido pela Equação (8). De acordo com o cálculo de $h(c_i)$ e de $L(c_{xy}, c_{pq})$, a pontuação de um agrupamento é definida em termos da quantidade máxima de relacionamentos de dependência internos a um agrupamento: a pontuação de um agrupamento é incrementada

para cada relacionamento existente entre seus elementos constituintes, e decrementada para aqueles que estão ausentes.

$$h(c_i) = \begin{cases} \sum_{a=1}^{k_{i-1}} \sum_{b=a+1}^{k_i} L(c_{ia}, c_{ib}) & , se \ k_i > 1 \\ 0, caso \ contrário \end{cases} \quad (7)$$

$$L(c_{xy}, c_{pq}) = \begin{cases} 1, se \ existe \ relacionamento \ entre \ c_{xy} \ e \ c_{pq} \\ -1, caso \ contrário \end{cases} \quad (8)$$

Tal como a função MQ original, a função EVM está baseada em grafos não-ponderados (DSM binária) e não considera o tamanho dos agrupamentos, o que consiste em severas limitações, conforme discutido anteriormente, significando que esta função não é adequada para os propósitos do presente trabalho.

Em todas as abordagens mencionadas até então, o problema de agrupamento de componentes de software é caracterizado como sendo um problema de único objetivo. Em [Praditwong et al. 2010] é então definida uma abordagem multi-objetivo baseada em busca pareto-ótima através de algoritmos genéticos. A abordagem avalia duas funções-objetivo (aqui referenciadas de “estratégia” pelo fato de representarem um conjunto de objetivos): MCA (*Maximizing Cluster Approach*) e ECA (*Equal-size Cluster Approach*).

A estratégia MCA tem o objetivo de maximizar a soma de conexões internas aos agrupamentos, minimizar a soma das conexões externas aos agrupamentos, maximizar o número de agrupamentos, maximizar o valor da função MQ e minimizar o número de agrupamentos isolados (com apenas um componente). A estratégia ECA difere da MCA pelo fato de ao invés de objetivar minimizar o número de agrupamentos isolados, objetiva minimizar a diferença entre o número máximo e mínimo de componentes em cada agrupamento. Como a definição dos objetivos é específica de cada problema, não é possível generalizar as referidas estratégias para o agrupamento de componentes em geral. No caso do presente trabalho, por exemplo, a estratégia MCA não é adequada pelo fato de considerar a função MQ, que já foi anteriormente discutida. Já a estratégia ECA sugere a uniformização do tamanho dos agrupamentos gerados, o que não é objetivado no presente trabalho. Uma solução com agrupamentos de tamanho aproximado pode não ser adequada na perspectiva da maximização das dependências internas aos agrupamentos e minimização das dependências entre agrupamentos.

Apesar disso, a idéia de estruturar o problema de agrupamento de componentes segundo uma abordagem multi-objetivo é relevante, já que uma vez que tais objetivos são adequadamente definidos, é possível obter bons resultados. A desvantagem é que problemas multi-objetivo em geral demandam mais esforço de desenvolvimento e maior custo computacional. Dessa forma, deve ser analisado se os prováveis ganhos na qualidade dos resultados produzidos compensam o custo envolvido. Além disso, a comparação dessa abordagem com outras de único objetivo é complexa, já que a forma de estruturar o problema difere em sua essência.

3.3.2. Estratégia de Busca

As estratégias de busca adotadas pelas abordagens discriminadas na Tabela 4 consistem em: Busca exaustiva, *Hill Climbing*, *Multiple Hill Climbing*, Algoritmos Genéticos e *Simulated Annealing*. A busca exaustiva verifica todas as soluções de agrupamento existentes para um dado problema, a fim de alcançar a solução ótima. De acordo com [Mancoridis et al. 1998], essa estratégia só é aplicável para um cenário com até 15 (quinze) componentes, devido ao elevado número de soluções.

A metaheurística *Hill Climbing*, apesar de ser aplicada em muitos experimentos com resultados bons em termos de qualidade e de custo de processamento, tal como em [Mancoridis et al. 1998], [Mancoridis et al. 1999], [Doval et al. 1999], [Harman et al. 2002], [Mitchell, Mancoridis 2002] e [Harman et al. 2005], tende a convergir prematuramente para um ótimo local e depende do ponto de partida da busca. Visando superar tais limitações, as demais estratégias são alternativamente adotadas.

A metaheurística denominada *Multiple Hill Climbing*, que é adotada em [Mahdavi et al. 2003], consiste numa variação da estratégia *Hill Climbing*, na qual a busca é repetida para uma mesma entrada sucessivas vezes, mudando apenas o ponto de partida da busca, a fim de aumentar a probabilidade de atingir a solução ótima. Após as várias repetições, uma avaliação comparativa é feita entre os resultados obtidos, e as características comuns aos melhores resultados (segundo algum critério de corte) são usadas para definir “blocos de construção” (alguns componentes são fixados em agrupamentos), os quais são utilizados como ponto de partida para novas execuções do algoritmo. Como o número de execuções é arbitrário, assim como o critério para definição das melhores soluções, é difícil generalizar essa estratégia para o problema de agrupamento de componentes. Além disso, o custo computacional envolvido é elevado.

A metaheurística *Simulated Annealing*, adotada no presente trabalho e também explorada em [Mitchell, Mancoridis 2002], tem se mostrado adequada no contexto de agrupamento de componentes de software por possibilitar que o espaço de busca seja melhor explorado, aumentando as chances do algoritmo escapar de ótimos locais e tornando a busca menos dependente do seu ponto de partida. Além disso, até o presente momento não há estudos que comprovem que está não é uma boa estratégia de busca para esse problema.

Algoritmos genéticos despendem maior custo computacional, e de acordo com [Mancoridis et al. 1998] e [Mancoridis et al. 1999], *Hill Climbing* consistentemente produz resultados de melhor qualidade. Dadas as limitações já discutidas desta última metaheurística, a eficiência de algoritmos genéticos torna-se realmente questionável.

3.4. Agrupamento de Elementos Aplicado à Engenharia Mecânica

O agrupamento de elementos no contexto da engenharia mecânica segue exatamente a mesma lógica empregada no contexto de reengenharia de sistemas de software, sendo até adotadas as mesmas estratégias de busca, com a vantagem de utilizar DSM, que é mais fácil de elaborar e compreender do que MDG, além de ser mais adequada para processamento computacional. A Tabela 5 a seguir resume os principais algoritmos de agrupamento de elementos aplicadas à engenharia mecânica que foram identificadas.

Conforme pode ser observado, as abordagens não lidam com desenvolvimento de produtos de software e diferem entre si em termos da unidade de agrupamento, o esquema de representação de dependência, a função objetivo e a estratégia de busca que adotam.

O estudo dessas abordagens levou à constatação de que, em geral, a estratégia de mensurar relacionamentos é pouco explorada, sendo dada maior ênfase ao algoritmo de agrupamento, o que na verdade também acontece nas abordagens aplicadas à MDG anteriormente discutidas. Assim, os valores de dependência usados em geral estão em uma escala qualitativa (por exemplo, “1” para baixa dependência e “2” para alta dependência) e cabe ao profissional construir a DSM com base na sua experiência, de maneira totalmente subjetiva. Abordagens mais elaboradas definem diferentes tipos de dependência, cada um com um valor de relevância associado, tal como em [Helmer et al. 2008]. Os valores, contudo, ainda são descrições qualitativas que são atribuídas de forma subjetiva.

Tabela 5. Algoritmos para agrupamento aplicados à engenharia mecânica.

	Unidade	Representação de Dependência	Função objetivo	Estratégia de busca
[Idicula 1995]	Tarefas	Valores qualitativos; 1 tipo	Custo de coordenação	<i>Hill Climbing</i>
[Fernandez 1998]	Equipes	Valores qualitativos; 1 tipo	Custo de coordenação	<i>Simulated Annealing</i>
[Thebeau 2001]	Equipes	Valores qualitativos; 1 tipo	Custo de coordenação	<i>Simulated Annealing</i>
[Whitfield et al. 2002]	Componentes físicos	Valores qualitativos; 1 tipo	Distância em relação à diagonal principal	Algoritmos genéticos
[Wang, Antonsson 2004]	Componentes físicos	Não especificado	Princípio MDL	Algoritmos genéticos
[Yu et al. 2007]	Equipes; Componentes físicos	Valores qualitativos; 1 tipo	Princípio MDL	Algoritmos genéticos
[Helmer et al. 2008]	Componentes físicos de produtos mecatrônicos	Valores qualitativos; 5 tipos (estrutural, de energia, de sinal, material e espacial)	Princípio MDL	Algoritmos genéticos

O algoritmo de [Idicula 1995] objetiva encontrar uma solução de agrupamento de atividades cujo custo de coordenação total seja consideravelmente reduzido. O algoritmo inicia considerando cada atividade individualmente como um agrupamento, e aleatoriamente escolhe uma atividade e calcula a afinidade desta com os demais agrupamentos. A atividade é então integrada ao agrupamento com o qual tem maior afinidade (uma nova solução é definida) e é calculado o custo total de coordenação. Havendo redução no custo de coordenação, a modificação proposta é aceita, caso contrário ela é rejeitada. Esse processo então continua até que, após inúmeras tentativas, não haja mais melhorias no custo de coordenação. Esse algoritmo tem a vantagem de não requerer que o tamanho e/ou a quantidade de agrupamentos seja previamente definida. Contudo, ele admite que uma atividade pertença a mais de um agrupamento, o que não é desejável no presente trabalho, dado que um componente deve ser desenvolvido por uma única equipe.

Em [Fernandez 1998], o algoritmo de [Idicula 1995] foi adaptado para lidar com o agrupamento de equipes e para melhorar a qualidade das soluções geradas. Assim, a função de custo total de coordenação foi modificada e foram introduzidos parâmetros para controlar como o algoritmo explora o espaço de busca e encontra uma solução. Por exemplo, para limitar o tamanho dos agrupamentos formados e com isso evitar que haja agrupamentos

desproporcionais. Além disso, foram introduzidos comportamentos aleatórios nas etapas do algoritmo através da técnica de *Simulated Annealing*, a fim de aumentar o espaço de solução explorado pelo algoritmo e minimizar a probabilidade deste resultar em uma solução sub-ótima. No entanto, experimentos posteriores mostraram que o algoritmo de [Fernandez 1998] nem sempre resulta na melhor solução identificada ao longo da busca.

O algoritmo de [Thebeau 2001] consiste numa reformulação do algoritmo de [Fernandez 1998], a fim reduzir consideravelmente a sobreposição entre agrupamentos, bem como melhorar a qualidade das soluções geradas. Para tanto, o algoritmo se baseia na técnica de *Simulated Annealing* para gerar uma nova solução, bem como para decidir se esta deve ser aceita, e também armazena a melhor solução que é encontrada ao longo da busca, a fim de evitar que por conta de SA, o resultado da busca consista em uma solução inferior a esta.

Para gerar uma nova solução, o algoritmo avalia o nível de dependência existente entre um dado componente escolhido randomicamente e os agrupamentos existentes, segundo o cálculo de uma *função de vínculo*. Em geral, o componente deve ser adicionado ao agrupamento com o qual possui maior vínculo. No entanto, por causa de SA, um componente pode vir a ser adicionado ao agrupamento com o qual possui o segundo maior vínculo. O problema dessa estratégia de gerar uma nova solução é que ela tende a restringir o espaço de busca, podendo levar o algoritmo a ficar preso a um ótimo local prematuramente. Experimentos realizados para comparar essa estratégia e a estratégia adotada no presente trabalho (seleção randômica do componente e do módulo) confirmam esse efeito, pois foram encontradas soluções melhores no segundo caso.

Já o cálculo da probabilidade de aceitação de uma solução inferior é feito através de uma equação diferente da tradicional *equação de Metropolis*, adotada pela maioria dos algoritmos de SA e também no presente trabalho. Experimentos realizados adotando essa nova equação evidenciaram que ela possibilita uma ampla aceitação de soluções inferiores, o que acaba por prejudicar a busca, fazendo o algoritmo resultar em soluções piores em comparação às soluções encontradas quando do uso de uma versão do algoritmo que utiliza a *equação de Metropolis*.

O algoritmo de [Whitfield et al. 2002] define uma função de agrupamento baseada nos valores de dependência entre componentes e na distância destes valores em relação à diagonal principal da DSM, e emprega algoritmos genéticos para encontrar a solução de agrupamento

que reduz consideravelmente o valor dessa função. Os agrupamentos gerados definem uma relação hierárquica entre os componentes, o que foge do escopo da abordagem proposta.

Em [Yu et al. 2003] e [Wang, Antonsson 2004] foi introduzido o princípio MDL (*Minimal Description Length*) como critério de agrupamento, fundamentado na teoria da informação. A idéia básica é desenvolver um modelo de descrição para DSM de maneira que a quantidade de dados utilizada seja significativamente pequena quando o modelo retrata uma arquitetura de maior grau de modularização. Então a melhor solução de agrupamento é encontrada através da busca pela minimização do valor de MDL realizada através de algoritmos genéticos. Em [Yu et al. 2003] somente é admitida DSM binária, que já se sabe que não caracteriza precisamente a dependência entre os elementos. Já em [Wang, Antonsson 2004], o foco está na definição de agrupamentos hierárquicos, igualmente ao algoritmo de [Whitfield et al. 2002], o que foge dos interesses do presente trabalho.

Em [Yu et al. 2007], o algoritmo de [Yu et al. 2003] é melhorado, a fim de ser utilizado com DSM numérica (com um único valor por célula, que retrata o relacionamento estrutural entre os componentes do produto) e permitir a intervenção humana nos agrupamentos sugeridos. Apesar disso, ambos os algoritmos requerem que seja especificado um número máximo de agrupamentos a serem gerados, o que não é adequado porque restringe a busca. Além disso, tais algoritmos também permitem que um elemento pertença a mais de um agrupamento, assim como o algoritmo proposto em [Idicula 1995].

Por fim, o algoritmo de [Helmer et al. 2008] consiste em uma melhoria do algoritmo de [Yu et al. 2007], no qual não é necessário especificar um número máximo de agrupamentos e são reduzidas as chances de haver componentes pertencentes a mais de um agrupamento. No entanto, conforme definido em [Helmer et al. 2008], esse algoritmo lida apenas com DSM numérica que contém informação sobre cinco tipos de dependência (estrutural, de energia, de sinal, material e espacial), não se mostrando adequado para lidar com produtos que não apresentam todos esses tipos de dependência, como é o caso de produtos de software.

Uma análise comparativa detalhada entre as abordagens não é possível, já que a qualidade das soluções de agrupamento não depende exclusivamente da estratégia de busca empregada, mas também da maneira como a dependência é quantificada, o que por sua vez depende do problema em questão.

3.5. Considerações Finais

A partir dos trabalhos apresentados neste capítulo, é possível perceber que o problema de agrupar elementos priorizando a maximização da dependência entre elementos em um mesmo agrupamento e a minimização da dependência entre elementos em agrupamentos distintos é um problema que se aplica em diferentes áreas do conhecimento (engenharia mecânica e engenharia de software), e cuja solução pode ser aplicada para diferentes propósitos.

No caso da engenharia de software, foram identificados trabalhos visando: refinar um projeto de arquitetura de domínio; minimizar a necessidade de comunicação entre equipes geograficamente distantes no contexto de DDS, tal como o presente trabalho, mas com ênfase na atividade de manutenção de software; e recuperar a arquitetura de um sistema. Para alcançar seus objetivos, esses trabalhos se baseiam em informações de dependência extraídas a partir do código de implementação, mas sem prover informação sobre como efetivamente realizar essa tarefa, diferentemente do presente trabalho, que avalia dependências em nível arquitetural e oferece métricas para esse fim. Além disso, os trabalhos analisados ou não especificam um modelo para representar dependências, ou adotam a representação em grafos denominada MDG. O uso de DSM para representar dependências entre elementos é predominante na engenharia mecânica, e gradativamente tem sido introduzido ao desenvolvimento de software.

Com exceção do trabalho apresentado em [Blois et al. 2005], todos os demais oferecem um algoritmo baseado em metaheurística para automatizar a geração de soluções de agrupamento. As metaheurísticas predominantemente usadas são *Hill Climbing*, Algoritmos Genéticos e *Simulated Annealing*. Dado que *Hill Climbing* consistentemente produz resultados de melhor qualidade em comparação à Algoritmos Genéticos [Mancoridis et al. 1998] [Mancoridis et al. 1999], ao mesmo tempo que *Hill Climbing* tende a convergir prematuramente para um ótimo local e depende do ponto de partida da busca, a metaheurística SA tem se mostrado mais adequada, sendo esta a metaheurística utilizada no presente trabalho. SA possibilita que o espaço de busca seja melhor explorado, aumentando as chances do algoritmo escapar de ótimos locais, e sem despender um elevado custo computacional.

Por fim, dentre os trabalhos apresentados, apenas o trabalho descrito em [Praditwong et al. 2010] estrutura o problema de agrupamento como sendo um problema multiobjetivo. Dessa forma, uma investigação mais ampla acerca dessa forma alternativa de estruturar o problema se mostra interessante.

Capítulo 4

Framework de Recomendação para Alocação de Equipes em Projetos Distribuídos de LPS

O presente trabalho compreende uma das etapas de um *framework* de recomendação para alocação de equipes em projetos distribuídos de linhas de produto de software, que visa auxiliar o gerente de projeto na tomada de decisão sobre como alocar equipes de desenvolvimento aos módulos de software. Para tanto, o *framework* provê um conjunto de recomendações de alocação que são geradas com base nos requisitos técnicos dos módulos a serem desenvolvidos e nos atributos técnicos e não-técnicos das equipes. Neste capítulo esse *framework* é apresentado em maiores detalhes.

4.1. Introdução

A alocação de equipes no desenvolvimento de software é uma tarefa complexa, geralmente deixada a cargo do gerente do projeto, para o qual falta o suporte adequado para analisar as inúmeras soluções possíveis dentro dos vários cenários de desenvolvimento de software existentes. Conforme observado em [Shen et al. 2002], processos para alocação de equipes geralmente se baseiam na capacidade técnica das equipes e nos requisitos das tarefas a serem desenvolvidas. Em se tratando de projetos de desenvolvimento distribuído, entretanto, essa estratégia não é suficiente, já que a separação geográfica entre as equipes quase sempre compromete a comunicação entre elas, o que por consequência acaba influenciando a qualidade do trabalho por elas desempenhado, causando o aumento do tempo e do custo de desenvolvimento dos projetos, dentre outros problemas.

Tomando por base o cenário de desenvolvimento distribuído de projetos de LPS, favorável para a distribuição da atividade de implementação de componentes de software entre equipes geograficamente dispersas, o *framework* descrito no presente capítulo consiste

em uma abordagem sistemática de recomendar soluções de alocação de equipes para este cenário.

A abordagem visa minimizar a necessidade de colaboração entre os diferentes lugares, uma vez que essa medida minimiza o impacto negativo dos problemas de comunicação e coordenação, prevenindo que equipes distintas sejam atribuídas a componentes fortemente dependentes. Além disso, a abordagem visa minimizar as diferenças entre as equipes que irão interagir mais intensamente através da redução das diferenças culturais e de fuso-horário entre elas, bem como reduzir a dificuldade de implementação fazendo a correspondência entre os requisitos dos componentes com o conhecimento e a experiência das equipes que lhes são atribuídas. Como resultado do uso dessa abordagem, é esperado um significativo aumento da qualidade dos projetos de DDS.

Para alcançar todos esses objetivos sem ignorar as particularidades de cada projeto, a abordagem adota o conceito de *framework*, significando que pode ser vista como uma orientação estruturante do processo de alocação de equipes, que deve ser instanciada e personalizada para cada projeto.

4.2. Fluxo de Atividades do Framework

Visando auxiliar o gerente de projeto na tomada de decisão sobre como alocar equipes de desenvolvimento aos componentes de software de um projeto de LPS, o *framework* consiste em quatro fases, conforme descrito pela Figura 3. Combinadas, essas fases produzem três tipos de recomendação e, também, oferecem *feedback* sobre o desempenho das equipes, que é útil para a geração de recomendações de alocação em projetos posteriores. As recomendações consistem em:

- Um conjunto de módulos de software $M = \{m_1, m_2, \dots, m_n\}$ que podem ser implementados de forma independente por equipes no conjunto $E = \{e_1, e_2, \dots, e_x\}$ de equipes que podem participar do projeto;
- Um mapeamento $EM = \{(m, ET) \mid m \in M \wedge ET \subseteq E\}$ de cada módulo de software m para o subconjunto de equipes ET que são tecnicamente habilitadas à implementar o módulo;
- Um conjunto de alocações $A = \{(m, e) \mid m \in M \wedge (m, ET) \in EM \wedge e \in ET\}$, onde cada alocação visa auxiliar o gerente do projeto a decidir sobre como atribuir cada equipe e à um módulo de software m .

Uma vez que a saída gerada pelo *framework* consiste em recomendações, significa que estas não são soluções exatas, estando passíveis de serem adaptadas pelo gerente para atenderem às necessidades específicas de um projeto. Por exemplo, dentre o conjunto de módulos recomendado, pode haver algum módulo que seja considerado muito complexo para ser desenvolvido por uma única equipe e dessa forma, o gerente pode julgar necessário fragmentá-lo em dois módulos menos robustos. Uma modificação dessa natureza, por sua vez, gera a necessidade de se redefinir as recomendações de equipes e de alocação, significando que o *framework* como um todo deve ser reaplicado para o projeto em questão.

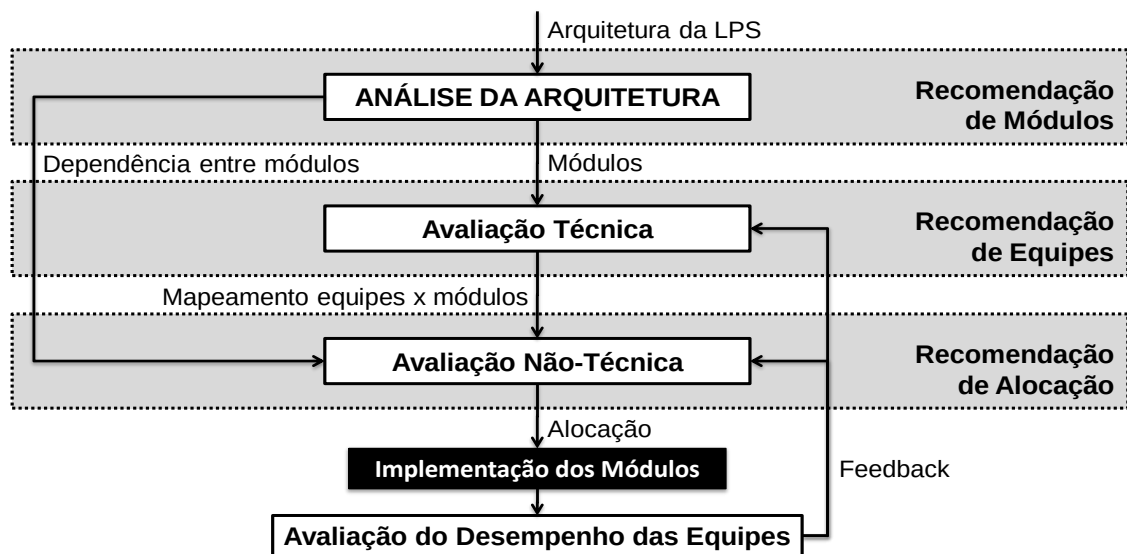


Figura 3. Fluxo de atividades do *framework*.

4.2.1. Análise da Arquitetura

A fase de *Análise da Arquitetura* corresponde ao presente trabalho, e por isso está destacada em letras maiúsculas na Figura 3, diferenciando-se das demais fases representadas. No presente capítulo esta fase é descrita de maneira breve, sendo detalhada no capítulo seguinte. Ela recebe como entrada a arquitetura baseada em componentes do projeto de LPS que, conforme anteriormente explicado, pode representar uma arquitetura de domínio ou uma arquitetura de aplicação. O propósito dessa fase é gerar módulos de software fracamente dependentes, e que por sua vez, correspondem ao agrupamento de componentes de software fortemente dependentes entre si. Nesse contexto, um módulo de software representa a unidade de trabalho que deve ser executada por uma equipe de desenvolvimento de software, ou seja, o conjunto $M = \{m_1, m_2, \dots, m_n\}$ de módulos de software que podem ser implementados de forma (parcialmente) independente por equipes diferentes.

A fim de alcançar seu objetivo, a fase de *Análise da Arquitetura* provê métricas arquiteturais para guiar o gerente do projeto na avaliação da dependência entre os componentes, bem como um algoritmo baseado na metaheurística *Simulated Annealing* para encontrar uma solução de agrupamento satisfatória, lidando com o agrupamento de componentes em módulos como um problema de otimização.

Por exemplo, dado um projeto hipotético com três componentes c_1 , c_2 e c_3 para serem desenvolvidos, há cinco soluções possíveis de agrupamento descritas pelos conjuntos de módulos M_1 , M_2 , M_3 , M_4 e M_5 apresentados a seguir.

- $M_1: m_1 = \{c_1\}, m_2 = \{c_2\}, m_3 = \{c_3\}$
- $M_2: m_1 = \{c_1, c_2\}, m_2 = \{c_3\}$
- $M_3: m_1 = \{c_1, c_3\}, m_2 = \{c_2\}$
- $M_4: m_1 = \{c_1\}, m_2 = \{c_2, c_3\}$
- $M_5: m_1 = \{c_1, c_2, c_3\}$

Cabe ao algoritmo de agrupamento avaliar as soluções (ou parte delas, pois na prática o universo de soluções é bem maior), com base na dependência entre os componentes, indicando uma boa solução de agrupamento, ou seja, uma solução que priorize o agrupamento de componentes fortemente dependentes entre si, de forma que a dependência entre módulos seja reduzida.

A partir da solução de agrupamento identificada pelo algoritmo, as dependências entre módulos são capturadas, a fim de serem consideradas na fase de *Avaliação Não-Técnica* para a geração de recomendações de alocação. A partir do nível de dependência entre os módulos é possível inferir sobre a necessidade de comunicação entre as equipes que serão alocadas para desenvolvê-los.

4.2.2. Avaliação Técnica

Uma vez que os módulos foram identificados, na fase de *Avaliação Técnica* estes devem ser caracterizados em termos dos requisitos técnicos necessários para implementá-los, assim como as equipes participantes $E = \{e_1, e_2, \dots, e_x\}$ devem ser caracterizadas em termos do conhecimento técnico requerido para implementar os módulos. Os requisitos técnicos dos módulos são identificados a partir da definição do nível de complexidade de desenvolvimento esperado para cada tecnologia que deve ser empregada (sistema operacional, plataformas, linguagens de programação, ferramentas, estruturas, processos, etc), e para o domínio de

conhecimento envolvido por cada módulo de software. Semelhantemente, as equipes são caracterizadas tecnicamente a partir da identificação do nível de conhecimento que esta possui nas tecnologias empregadas nos módulos e no domínio envolvido por eles, o que em geral depende das experiências de trabalho da equipe e da formação acadêmica de seus integrantes.

A caracterização técnica dos módulos e das equipes são usadas em conjunto com algoritmos, regras e políticas baseadas em lógica nebulosa (lógica *fuzzy*) para recomendar o mapeamento $EM = \{(m, ET) \mid m \in M \wedge ET \subseteq E\}$, associando cada módulo de software m ao subconjunto de equipes ET que são tecnicamente habilitadas para implementá-los. Utilizando os termos difusos, o gerente de projeto pode alterar o conjunto de regras, refinando as políticas para atenderem às necessidades específicas de cada projeto. Por exemplo, se a política da organização é poupar equipes mais experientes para trabalhos mais difíceis, o processo de avaliação definirá como adequadas as equipes que possuírem conhecimentos mais próximos aos demandados pelos módulos. Por outro lado, se o gerente deseja finalizar o projeto o mais rápido possível, o conjunto de regras é alterado de forma que equipes com qualificações mais altas sejam sempre mais adequadas.

4.2.3. Avaliação Não-Técnica

Na fase de *Avaliação Não-Técnica* são geradas as recomendações de alocação $A = \{(m, e) \mid m \in M \wedge (m, ET) \in EM \wedge e \in ET\}$ que atribui cada equipe específica e a um módulo de software m , com base em informações sobre os módulos, as equipes e os fatores decorrentes da separação geográfica entre elas que influem na comunicação. A idéia é que os módulos possam ser desenvolvidos da forma mais independente possível dos outros módulos, que equipes tecnicamente qualificadas sejam alocadas aos módulos e, quando for necessário ocorrer interação entre as equipes, que isto seja feito da melhor forma possível.

Por exemplo, se módulos dependentes entre si são alocados às equipes A e B tecnicamente habilitadas para implementá-los, é pressuposto que haverá maior necessidade de comunicação entre tais equipes. Assim, é preciso verificar a capacidade destas em estabelecer comunicação efetiva, conforme seus atributos não-técnicos. Se a equipe A só pode se comunicar em inglês e a equipe B apenas em espanhol, tais equipes provavelmente enfrentarão bastante dificuldade de comunicação, já que precisarão aprender um novo idioma em curto prazo. No entanto, se a equipe A também conhece a língua espanhola, embora apenas em termos de leitura e escrita, as equipes podem estabelecer comunicação com relativa facilidade através de *chat* e *e-mail*. Nessa situação, a viabilidade da comunicação é então

definida pela disponibilidade das equipes em termos de fuso-horário e horário de trabalho, já que *chat* exige que as equipes estejam conectadas no mesmo instante, ao contrário de *e-mail*.

Dessa forma, na fase de *Avaliação Não-Técnica*, inicialmente as equipes são caracterizadas segundo seus atributos não-técnicos, tais como localização, língua e disponibilidade, dentre outros atributos relevantes no contexto de DDS. A partir de tais atributos é possível estimar o grau de dificuldade que as equipes podem enfrentar para estabelecer comunicação entre si. Em seguida, com base na *dependência entre módulos*, identificada na fase de *Análise da Arquitetura*, é estimada a necessidade de comunicação entre as equipes que irão implementar cada módulo. Então, a partir da utilização de algoritmos genéticos, tais informações são associadas ao *mapeamento equipes x módulos* derivado da fase de *Análise Técnica*, a fim de gerar as recomendações de alocação, admitindo que dependendo do número de equipes e de módulos para serem implementados, uma mesma equipe possa ser alocada a mais de um módulo.

4.2.4. Avaliação do Desempenho das Equipes

Após a conclusão da implementação do projeto, a fase de *Avaliação do Desempenho das Equipes* deve qualificar a performance das equipes alocadas, tanto em termos técnicos, verificando se o trabalho que lhes foi designado foi realizado adequadamente (*reputação*), quanto em termos não técnicos, avaliando como se sucedeu o relacionamento entre as equipes (*afinidade*). Isso é feito por meio da aplicação de questionários destinados às equipes e ao gerente do projeto.

Além disso, os requisitos técnicos definidos para os módulos durante a fase de *Avaliação Técnica* também são avaliados, a fim de verificar a existência de diferenças entre o que foi planejado e o que foi efetivamente utilizado. Por exemplo, o arquiteto do projeto pode ter definido uma determinada tecnologia para ser empregada no desenvolvimento de um módulo e a equipe responsável por desenvolver o módulo pode ter sentido a necessidade de utilizar outra tecnologia. O *framework* deve manter um histórico das tecnologias empregadas para facilitar o trabalho dos desenvolvedores quando da integração dos módulos desenvolvidos.

Com base nessa avaliação, a informação de *feedback* é derivada para refinar as caracterizações técnicas e não-técnicas das equipes participantes e a caracterização técnica dos módulos, a fim de melhorar a qualidade das recomendações geradas pelo *framework* em projetos futuros.

4.3. Considerações Finais

O *framework* foi projetado visando oferecer flexibilidade, a fim de ser instanciado com facilidade em diferentes projetos distribuídos de LPS. Para tanto, ele foi organizado em fases independentes, com entradas e saídas bem definidas, o que permite a evolução das fases ou mesmo a modificação das técnicas e algoritmos empregados em cada uma delas, sem exercer qualquer impacto sobre as demais. Além disso, ainda é possível adicionar novas fases, como por exemplo, uma fase para avaliar aspectos de negócio, incluindo tempo e custo de desenvolvimento.

A vantagem da organização do *framework* em fases não se limita ao aspecto de flexibilidade. As fases possibilitam a organização das recomendações de alocação em três tipos, proporcionando ao gerente do projeto um melhor entendimento sobre as relações causais entre as variáveis avaliadas, referentes aos módulos de software e aos atributos técnicos e não-técnicos das equipes.

Dentre as fases compreendidas pelo *framework*, a fase de *análise da arquitetura*, objeto de estudo do presente trabalho, mostra-se de significativa contribuição na literatura específica de DDS. Abordagens existentes correlatas ao *framework* apresentado, tais como [Setamanit et al. 2007; Sooraj, Mohapatra 2008; Lamersdorf, Munch 2009], em geral não se preocupam em modelar as atividades de desenvolvimento de forma favorável à distribuição entre equipes remotas. Elas simplesmente tomam como entrada as atividades e definem estratégias para identificar equipes habilitadas tecnicamente para desenvolvê-las, e ao mesmo tempo, facilitar a interação entre tais equipes com base em seus atributos não-técnicos.

Capítulo 5

Abordagem para Recomendação de Módulos para Projetos Distribuídos de LPS

A abordagem para recomendação de módulos para projetos distribuídos de LPS compreende a etapa de *análise da arquitetura do framework* de recomendação para alocação de equipes apresentado no Capítulo 4. Conforme explicado nesse capítulo, a abordagem possui três objetivos fundamentais:

- Mensurar o grau de dependência em nível arquitetural entre componentes de software de um projeto de LPS;
- Identificar módulos de software que possam ser implementados e integrados por equipes diferentes (unidades mínimas de trabalho de implementação) com reduzida necessidade de comunicação, a partir do agrupamento de componentes dependentes;
- Mensurar o grau de dependência entre módulos.

A fim de alcançar os objetivos especificados, a abordagem está organizada em três etapas. São elas: *medição de dependências entre componentes*, *recomendação de módulos* e *medição de dependências entre módulos*. A interação entre essas etapas é ilustrada na Figura 4 a seguir.

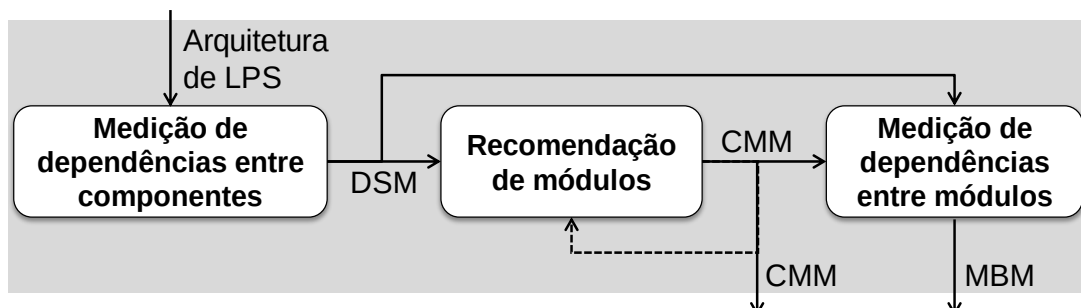


Figura 4. Organização da abordagem proposta.

A etapa de *medição de dependências entre componentes* lida com a tarefa de orientar o engenheiro de software no projeto de uma DSM, que deve refletir a estrutura do projeto com

base nas dependências entre componentes de software. Nesse sentido, essa etapa define métricas arquiteturais que visam tornar a avaliação da dependência entre componentes uma atividade menos subjetiva, e que devem ser aplicadas de forma a respeitar as propriedades específicas de LPS. O resultado dessa etapa consiste na construção da DSM do projeto, que reflete uma iteração de desenvolvimento de componentes. Um exemplo de DSM para um projeto é descrito pela Figura 5, a partir do qual é possível observar que a diagonal principal da matriz é sempre zero, e os valores nas demais células estão no intervalo fechado $[0,1]$. Nesse exemplo, a dependência do componente c_1 em relação ao componente c_3 é 1,00.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
c_0	0,00	0,11	0,47	0,11	0,17	0,11	0,17	0,11	0,17
c_1	0,11	0,00	0,11	0,44	0,17	0,11	0,17	0,11	0,17
c_2	1,00	0,11	0,00	0,11	0,17	0,11	0,17	0,11	0,39
c_3	0,11	1,00	0,11	0,00	0,17	0,11	0,17	0,11	0,39
c_4	0,17	0,17	0,17	0,17	0,00	0,17	0,33	0,17	0,33
c_5	0,11	0,11	0,11	0,11	0,17	0,00	0,17	0,33	0,17
c_6	0,17	0,17	0,17	0,17	1,00	0,17	0,00	0,17	0,33
c_7	0,11	0,11	0,11	0,11	0,17	1,00	0,17	0,00	0,17
c_8	0,17	0,17	0,43	0,39	0,33	0,17	0,56	0,33	0,00

Figura 5. Exemplo de DSM para um projeto.

Durante a etapa de *recomendação de módulos* é empregado um algoritmo de agrupamento sobre a DSM resultante da etapa anterior, a fim de gerar módulos de componentes. Algoritmos de agrupamento para DSM em geral operam segundo a definição de uma *função objetivo* e de uma *estratégia de busca* de solução. A função objetivo expressa o “custo” de uma configuração de componentes, e a estratégia de busca visa encontrar o menor valor possível para essa função. Como esse problema é Np-completo, os algoritmos buscam encontrar soluções próximas à solução ótima, podendo eventualmente alcançá-la. Como resultado, é gerado uma matriz denominada CMM (*Component-Module Matrix*), que identifica os componentes em cada módulo. Um exemplo de CMM é descrito pela Figura 6. Nesse exemplo, há 12 componentes e 3 módulos. Os componentes em cada módulo são identificados por “1”.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
M_0	0	0	0	0	1	0	1	0	1	1	1	1
M_1	0	1	0	1	0	1	0	1	0	0	0	0
M_2	1	0	1	0	0	0	0	0	0	0	0	0

Figura 6. Exemplo de CMM para um projeto.

Uma vez que linhas de produto podem ser implementadas em várias iterações de desenvolvimento, por exemplo, uma iteração só para os componentes da arquitetura de domínio e uma iteração única para os componentes da arquitetura de uma aplicação, o algoritmo de agrupamento deve gerar módulos considerando os módulos previamente definidos (se houver), admitindo que estes possam ser atualizados para envolver novos componentes (a seta tracejada na Figura 4 retrata esta situação).

Por fim, a etapa de *medição de dependência entre módulos* lida com a tarefa de orientar o engenheiro de software no projeto de uma matriz semelhante à DSM, cujas células na diagonal principal representam a dependência interna dos módulos de software e as demais células representam a dependência entre pares de módulos, conforme ilustrado pelo exemplo da Figura 7. A matriz resultante, denominada MBM (*Module Based Matrix*), deve ser usada para auxiliar o gerente do projeto na alocação de equipes de desenvolvimento aos módulos. A partir da dependência entre módulos é possível inferir sobre a necessidade de comunicação entre as equipes que irão implementá-los. Dessa forma, o gerente deve priorizar soluções de alocação em que as equipes que interajam mais apresentem menores diferenças em termos de cultura, fuso-horário, horário de trabalho e recursos, dentre outros fatores que dificultam a comunicação entre elas.

	M_0	M_1	M_2	M_3
M_0	1,00	0,00	0,06	0,12
M_1	0,60	0,28	0,00	0,00
M_2	0,00	0,00	0,18	0,00
M_3	0,00	0,04	0,00	0,00

Figura 7. Exemplo de MBM para um projeto.

As subseções 5.1, 5.2 e 5.3 a seguir apresentam em detalhes cada uma das etapas que compõe a abordagem.

5.1. Medição de Dependências entre Componentes

Componentes oferecem funcionalidades de sistema através da interação, cooperação e coordenação entre eles, o que tende a produzir dependências, em diferentes extensões. Conceitualmente, ocorre dependência quando um dado componente depende de outro(s) para realizar suas operações ou quando mudanças em um componente devem ser refletidas em outro(s) componente(s) [Vieira 2004].

Dependências entre componentes podem ser identificadas com base nos conceitos de coesão e acoplamento [Blois et al. 2005]. Componentes coesos possuem funcionalidade bem

definida, enquanto que componentes acoplados tendem a depender bastante um dos outros. Assim, é possível perceber que esses conceitos estão diretamente relacionados: componentes com alto nível de coesão tendem a ser menos acoplados, já que por estarem bem focados em uma funcionalidade, tendem a depender menos de outros componentes.

Com base em tais conceitos, a etapa de *medição de dependência entre componentes* define três métricas para capturar a dependência em nível arquitetural entre componentes: *dependência por interfaces*, *dependência por operações* e *dependência por features*. As duas primeiras métricas estão mais relacionadas ao conceito de acoplamento entre componentes, avaliando o quanto um componente precisa de outro, enquanto a última está mais relacionada ao conceito de coesão, avaliando a semelhança entre a funcionalidade dos componentes. Essas métricas são apresentadas em detalhes a seguir, e a demonstração de uso delas é feita no Capítulo 6, em que são apresentados os estudos de caso desenvolvidos.

Dependência por interfaces (DI). A métrica de dependência por interfaces expressa a dependência entre dois componentes com base nas interfaces que um componente requer do outro. A Equação (9) define a dependência por interfaces $DI_{A,B}$ do componente A para o componente B , na qual $IP_{B,A}$ representa o conjunto de interfaces que o componente B provê e o componente A requer, e IR_A representa o conjunto de todas as interfaces que o componente A requer. Trata-se de uma medida assimétrica, o que quer dizer que $DI_{A,B} \neq DI_{B,A}$.

$$DI_{A,B} = \begin{cases} \frac{|IP_{B,A}|}{|IR_A|}, & \text{se } IR_A \neq \emptyset \\ 0, & \text{caso contrário} \end{cases} \quad (9)$$

Dependência por operações (DO). A métrica de dependência por operações expressa a dependência entre dois componentes com base nas operações que um componente requer do outro. A Equação (10) define a dependência por operações $DO_{A,B}$ do componente A para o componente B , na qual $OP_{B,A}$ representa o conjunto de operações que o componente B provê e o componente A requer, e OR_A representa o conjunto de todas as operações que o componente A requer. Novamente, trata-se de uma medida assimétrica, o que significa que $DO_{A,B} \neq DO_{B,A}$.

$$DO_{A,B} = \begin{cases} \frac{|OP_{B,A}|}{|OR_A|}, & \text{se } OR_A \neq \emptyset \\ 0, & \text{caso contrário} \end{cases} \quad (10)$$

O resultado das métricas DI e DO está no intervalo fechado $[0,1]$, e deve ser admitido que se o componente B não provê interfaces ou operações, não existe dependência de A para

B e, portanto, o valor de $DI_{A,B}$ e de $DO_{A,B}$ é 0 (zero). Quanto mais próximo de 1 (um) é o valor da métrica, maior é a dependência do componente A em relação ao componente B.

Numa arquitetura de LPS, o valor dessas métricas tende a variar consideravelmente entre pares de componentes, uma vez que componentes obrigatórios podem prover interfaces ou operações que são requeridas apenas por componentes opcionais, por exemplo. Além disso, essas métricas não são equivalentes, e devem ser usadas alternadamente. Na verdade, a métrica DO é mais precisa do que a métrica DI, uma vez que se uma interface é requerida por um componente, ao menos uma operação nesta interface é invocada pelo componente. Por outro lado, é mais trabalhoso lidar com as operações do que com as interfaces, já que é necessário não apenas checar o diagrama de componentes do projeto e a documentação das interfaces, mas também os diagramas de sequência ou qualquer outro artefato que informe sobre o uso das operações.

Dependência por *features* (DF). A métrica de dependência por *features* define a dependência entre dois componentes com base nas *features* em comum por eles implementadas, segundo o mapeamento de *features* para componentes realizado durante a elaboração do projeto arquitetural. Essa medida consiste no cálculo do *índice de Jaccard* [Jaccard 1901], que foi matematicamente provado como sendo uma expressão de similaridade entre conjuntos. A Equação (11) define a dependência por *features* $DF_{A,B}$ entre os componentes A e B, na qual F_A and F_B representam o conjunto de *features* implementadas pelos componentes A e B, respectivamente. Trata-se de uma medida simétrica, ou seja, $DF_{A,B} = DF_{B,A}$.

$$DF_{A,B} = \frac{|F_A \cap F_B|}{|F_A \cup F_B|} \quad (11)$$

O resultado da equação também está no intervalo fechado $[0,1]$, onde quão mais próximo de 1 (um) é o valor, maior é a semelhança funcional ou afinidade entre os componentes A e B. Observe que como componentes sempre envolvem alguma *feature*, o denominador na equação nunca pode ser 0 (zero).

Diferentemente das métricas DI e DO, a métrica DF define a dependência entre dois componentes segundo o grau de afinidade entre eles. Nesse sentido, alguém poderia questionar que afinidade não implica em dependência ou não é sinônimo de dependência. De fato, componentes que implementam as mesmas *features* não necessariamente mantêm relação através de interfaces (ou operações) providas e requeridas, ou seja, nem sempre são dependentes estruturalmente. No entanto, componentes que implementam as mesmas *features*

desempenham funcionalidades correlatas, o que pode influenciar na intensidade de comunicação entre suas respectivas equipes de desenvolvimento, conforme observado em [Grinter et al. 1999]. Nesse estudo, foi comprovado que a separação de componentes que implementam uma mesma *feature*, fenômeno referenciado como *problema da distribuição de features*, contribui para tornar o projeto vulnerável à problemas de comunicação. Por essa razão, a métrica DF é tratada nesse trabalho como sendo uma medida de dependência indireta, tendo impacto inferior sobre a dependência entre componentes do que as métricas DI e DO.

5.1.1. Redução de Perspectiva

Conforme discutido, as métricas DI e DO devem ser usadas alternativamente. Já a métrica DF deve ser usada em conjunto com as demais, resultando na construção de uma DSM multidimensional, para a qual se faz necessário definir um processo de redução de perspectiva (*perspective reduction*), a fim de transformá-la em unidimensional.

Na presente abordagem, o processo de redução de perspectiva consiste no cálculo da média ponderada entre os valores das métricas utilizadas, conforme descrito a seguir. A Equação (12) é aplicada quando a métrica DI é usada em conjunto com a métrica DF, enquanto a Equação (13) é aplicada quando do uso das métricas DO e DF. Em ambas as equações, $D_{A,B}$ significa a dependência do componente A em relação ao componente B , α especifica pesos complementares para as métricas DI, DO e DF, e cujo valor deve ser ajustado para cada projeto de acordo com os interesses envolvidos. O valor $\alpha = 2/3$ é sugerido como valor *default* com base em vários experimentos conduzidos. Além disso, também é sugerido adotar $\alpha > 1/2$, porque o impacto das métricas DI e DO sobre a dependência é superior ao da métrica DF, conforme anteriormente discutido.

$$D_{A,B} = [DI_{A,B} \times \alpha] + [DF_{A,B} \times (1 - \alpha)], \quad 0 \leq \alpha \leq 1 \quad (12)$$

$$D_{A,B} = [DO_{A,B} \times \alpha] + [DF_{A,B} \times (1 - \alpha)], \quad 0 \leq \alpha \leq 1 \quad (13)$$

5.1.2. Construção da DSM

Nas subseções anteriores foram apresentadas métricas para avaliar a dependência entre componentes de software e um processo de redução de perspectiva para combiná-las, possibilitando a construção de uma DSM unidimensional. Nesta subseção são então feitas

considerações sobre a construção da DSM propriamente dita, dados os aspectos de *variabilidade e opcionalidade* de linhas de produto.

Idealmente a DSM de um projeto iria conter componentes da arquitetura de domínio ou componentes da arquitetura de uma aplicação. Contudo, na prática nem sempre é possível fazer uma separação rígida entre domínio e aplicação. Durante o projeto de uma LPS, por exemplo, um componente pode ser identificado e especificado como sendo parte da arquitetura do domínio e sua implementação real pode ser específica de produto, significando que esta será realizada apenas na fase de Engenharia de Aplicação. Em outra situação, a variação entre as aplicações pode ser tamanha, que não seja viável incorporá-la em uma única implementação do componente. Nesse caso, também devem ser desenvolvidas múltiplas implementações, com a diferença de que tais implementações podem ser desenvolvidas tanto na Engenharia de Domínio quanto na Engenharia de Aplicação.

Dado que a estratégia de implementação de componentes a ser adotada depende do planejamento e dos interesses envolvidos em cada projeto, não se faz adequado fixá-la no presente trabalho. Partindo disso, é admitido que sejam construídas inúmeras DSM para um mesmo projeto que, na verdade, ao invés de representarem a arquitetura de domínio ou a arquitetura de uma aplicação, representam uma iteração de desenvolvimento de componentes.

A DSM que caracteriza a primeira iteração de desenvolvimento de um projeto deve conter todos os componentes que serão desenvolvidos nessa iteração. Já a DSM das iterações subsequentes devem conter, além dos componentes que serão desenvolvidos, os módulos previamente definidos segundo os seus componentes constituintes. Isso previne que relações de dependência sejam omitidas, já que o fato de componentes serem desenvolvidos em iterações diferentes não significa que estes não estão relacionados.

Em LPS, a identificação dos componentes para serem desenvolvidos requer atenção no caso de ser utilizada a técnica de variabilidade por substituição, em que há uma especificação genérica do componente e múltiplas implementações possíveis (componentes variantes). Cada componente variante deve ser incluído individualmente na DSM. Caso contrário, supondo que componentes variantes mantenham as mesmas propriedades estruturais, uma vez que estão baseados em uma mesma especificação, o projetista da DSM pode achar adequado representá-los como um único componente, abstraindo suas diferenças. Como resultado, os componentes variantes são implicitamente agrupados juntos, o que na prática pode não ser a melhor solução. Conforme apresentado no Capítulo 6 através dos estudos de caso realizados,

componentes variantes geralmente implementam *features* diferentes, e até podem definir comportamentos diferentes pela habilitação ou desabilitação de interfaces e operações, o que pode resultar em valores diferentes para as métricas.

Por exemplo, na linha de produto de microondas descrita em [Gomaa 2004], comentada no Capítulo 2 e apresentada em detalhes no Capítulo 6, há o componente genérico *Display Interface*, que possui duas implementações distintas (*One-Line Microwave Display* e *Multi-line Microwave Display*). O projetista da DSM dessa linha de produtos deve incluir os componentes *One-Line Microwave Display* e *Multi-line Microwave Display* individualmente na DSM, e não simplesmente o componente genérico *Display Interface*.

Além disso, também é preciso atenção ao lidar com componentes opcionais. Componentes opcionais fazem parte da arquitetura do domínio e, portanto, são identificados durante a fase de Engenharia de Domínio. Apesar disso, conforme anteriormente discutido, isso não quer dizer que tais componentes só possam ser implementados nessa fase. Dessa forma, quando do uso da presente abordagem, deve ser definido em que iteração os componentes opcionais serão desenvolvidos, o que varia para cada projeto. Por exemplo, para um dado projeto pode ser definido que serão realizadas duas iterações de desenvolvimento, sendo a primeira para desenvolver componentes obrigatórios e a segunda para desenvolver componentes opcionais. Isso significa que a primeira DSM do projeto irá conter apenas componentes obrigatórios e a segunda DSM irá conter os componentes obrigatórios (que já foram agrupados) e os componentes opcionais, que poderão ser incluídos em algum módulo previamente definido ou poderão ser organizados em novos módulos. Já em um projeto cujo número de componentes opcionais é muito reduzido, pode ser definido que os componentes obrigatórios e opcionais serão desenvolvidos em uma única iteração.

5.2. Recomendação de Módulos

O problema de agrupar componentes em módulos pode ser estruturado como um problema de otimização combinatória, que consiste em encontrar as possíveis partições de um conjunto de elementos e, com base em um critério de avaliação (função objetivo), definir qual é a partição mais adequada (achar o melhor valor para a função objetivo). Dessa forma, uma partição de um conjunto X consiste em uma divisão de X em subconjuntos não-vazios e não-sobrepostos na qual cada elemento x em X pertence à exatamente um desses subconjuntos. De maneira formal, temos que P é uma partição de X se:

1. $\cup P = X$
2. $A \cap B = \emptyset$, se $A \in P, B \in P, A \neq B$

Por exemplo, dado $X = \{4, 5, 6\}$, as possíveis partições de X são: $\{\{4\}, \{5\}, \{6\}\}$; $\{\{4, 5\}, \{6\}\}$; $\{\{4, 6\}, \{5\}\}$; $\{\{4\}, \{5, 6\}\}$ e $\{\{4, 5, 6\}\}$.

O cálculo do número de partições possíveis para um dado conjunto com n elementos consiste no *número de Bell* [Rota 1964]. Existem diferentes equações para calcular o número de Bell. No presente trabalho, o cálculo é demonstrado pelo somatório do *número de Stirling do segundo tipo* [Abramowitz, Stegun 1972], conforme descrito pela Equação (14), onde k é o número de subconjuntos em uma partição. O cálculo do *número de Stirling do segundo tipo* é apresentado pela Equação (15) e expressa o número de formas de dividir um conjunto com n elementos em k subconjuntos não-vazios. De acordo com essa equação, quando há um único subconjunto, existe uma única forma de dividir o conjunto, que seria manter todos os elementos no mesmo subconjunto. Igualmente, se o número de elementos corresponde ao total de subconjuntos, cada subconjunto só contém um elemento e, portanto, também há uma única forma de dividir os elementos. Em todos os demais casos, o total de partições possível é definido por uma equação recursiva que depende apenas do total de elementos n e do número de subconjuntos k definido para uma partição.

$$B_n = \sum_{k=0}^n S(n, k)$$

$$S(n, k) = \begin{cases} 0, & \text{se } k = 0 \\ 1, & \text{se } k = 1 \text{ ou } k = n \\ S(n-1, k-1) + [k \times S(n-1, k)], & \text{demais casos} \end{cases} \quad (15)$$

Por exemplo, num conjunto com 3 elementos como no exemplo anteriormente (14) descrito, temos: $B_3 = S(3,0) + S(3,1) + S(3,2) + S(3,3) = 0 + S(2,0) + S(2,1) + S(2,1) + [2 \times S(2,2)] + 1 = 0 + 0 + S(1,0) + S(1,1) + S(1,0) + S(1,1) + 2 + 1 = 0 + 0 + 0 + 1 + 0 + 1 + 2 + 1 = 5$. Já para um conjunto com 10 elementos, esse valor aumenta para 115.975, enquanto que para 20 elementos, corresponde à 51.724.158.235.372.

Dado que o número de partições cresce rapidamente à medida que o tamanho do conjunto aumenta, fica claro que não é possível gerar e avaliar todas as partições para

qualquer conjunto. Uma estratégia para encontrar uma partição adequada com reduzido custo computacional é o uso de metaheurísticas, introduzidas no Capítulo 2.

O algoritmo para agrupamento de componentes da presente abordagem está baseado na metaheurística *Simulated Annealing*, na qual temos a seguinte analogia entre os termos usados no processo da termodinâmica e os termos usados em um problema de otimização combinatória: o nível de energia do metal consiste na função objetivo, para a qual se deseja encontrar o menor valor possível; a temperatura é o parâmetro de controle da busca, a partir da qual é definida a probabilidade de aceitação de uma solução pior; as variações de estado correspondem às soluções vizinhas; e o estado de solidificação do metal em configuração de mínima energia corresponde à solução ótima.

As subseções a seguir apresentam esse algoritmo em detalhes. A subseção 4.2.1 apresenta as premissas que inspiraram a definição da função objetivo do algoritmo. A subseção 4.2.2 explica as equações que definem a função objetivo do algoritmo, relacionando-as com as premissas. Por fim, na subseção 4.2.3 é apresentado o funcionamento geral do algoritmo.

5.2.1. Premissas

Premissa 1. *A intensidade de comunicação entre desenvolvedores é diretamente proporcional à intensidade das dependências entre os componentes de software por eles desenvolvidos.*

Essa é a premissa fundamental para o desenvolvimento da presente abordagem. Vários estudos atestam essa premissa: em [Grinter 2003] e em [De Souza et al. 2004] é comprovado que dependências entre pedaços de código criam dependências sociais, e em [Sosa, Eppinger et al. 2002] é identificada uma forte correlação entre entidades de software dependentes em um sistema e a frequência de comunicação entre as equipes que lidam com tais componentes. Por fim, conforme evidenciado em [Ghezzi et al. 2003], quanto mais independentes são os módulos, mais independentemente os desenvolvedores podem proceder com seus trabalhos.

Premissa 2. *A comunicação entre membros de equipes distintas tende a ser mais dispendiosa do que entre membros de uma mesma equipe, em particular se as equipes estiverem geograficamente distantes².*

² No contexto da presente abordagem, é admitido que os membros de uma equipe estão co-localizados, ou seja, compartilham um mesmo ambiente de trabalho.

A comunicação se apresenta como um dos grandes desafios para a realização de trabalho em grupo devido às diferenças entre as pessoas, não apenas em termos de conhecimento técnico (formação educacional, experiência no domínio do projeto, experiência com plataformas e linguagens de programação), mas de personalidade (temperamento, criatividade, inteligência, valores, dentre outras características psicológicas que determinam os padrões de pensar, sentir e agir do indivíduo). Apesar disso, uma vez que os membros de cada equipe estão co-localizados, estes compartilham o mesmo contexto organizacional e cultural, além de terem maiores possibilidades de conversas informais (não planejadas), fatores que juntos facilitam a comunicação. No entanto, a separação física inerente ao DDS dificulta ainda mais a comunicação em decorrência das diferenças em termos de disponibilidade, recursos (infra-estrutura) e valores sócio-culturais que (possivelmente) passam a existir entre as pessoas nos diferentes locais [Herbsleb 2007].

Premissa 3. *A intensidade de comunicação entre desenvolvedores é influenciada pela quantidade de componentes a serem desenvolvidos.*

Considerando que a implementação de componentes é, sobretudo, uma tarefa de desenvolvimento, o volume de trabalho a ser realizado pode ser definido em função da quantidade de componentes a serem desenvolvidos. Nesse sentido, a referida premissa relaciona a intensidade de comunicação entre desenvolvedores com o volume de trabalho a ser realizado em conjunto por eles, independentemente de estarem ou não na mesma equipe. Considerando que a complexidade de uma tarefa aumenta com o seu tamanho [Espinosa et al. 2007], e que tarefas complexas exigem maior interação entre as equipes responsáveis por desenvolvê-las [Silva 2008], é possível concluir que quanto maior o volume de trabalho a ser realizado em conjunto, maior tende a ser a intensidade de comunicação entre os desenvolvedores, em geral.

Além disso, é objetivo da presente estratégia prevenir duas situações extremas: módulos de tamanho máximo (todos os componentes em um único módulo) e módulos de tamanho mínimo (cada componente isolado em um módulo). O primeiro caso seria, teoricamente, a configuração ideal, na qual a dependência interna ao módulo é máxima e a dependência externa, mínima. Contudo, dado que módulos devem ser desenvolvidos por uma única equipe, esse resultado impossibilitaria o desenvolvimento distribuído. O segundo caso seria a pior configuração, na qual a dependência externa aos módulos é máxima e a

dependência interna, mínima, o que iria contra as premissas 1 e 2, e invalidaria o objetivo fundamental da estratégia de agrupar componentes em módulos.

5.2.2. Função Objetivo

O algoritmo iterativamente tenta diminuir o valor da função de *custo total de comunicação* (CTC), que pode ser entendida como uma medida da necessidade de comunicação em um projeto. O valor de CTC é calculado como mostrado pela Equação (16), onde M é o módulo que contém os componentes c_i e c_j , t é o total de componentes em M (*tamanho do módulo*), DSM_{c_j, c_i} é o valor da dependência do componente c_i em relação ao componente c_j , n é o número de componentes na DSM e $CTC_{\max}$ é o valor máximo possível de CTC . O valor de $CTC_{\max}$ corresponde ao custo da solução em que todos os componentes estão isolados em módulos distintos, situação em que todas as dependências são caracterizadas como dependências externas aos módulos, ou em que todos os componentes estão agrupados em um único módulo.

$$CTC = \begin{cases} CTC_{\max}, & \text{se } \exists c_i, c_j \in M, \text{ sendo } 0 \leq i, j < t, i \neq j \mid DSM_{c_j, c_i} = 0, DSM_{c_i, c_j} = 0 \\ \sum_{i=1}^n CC_{c_i}, & \text{caso contrário} \end{cases} \quad (16)$$

De acordo com a Equação (16), se a solução de agrupamento possui algum módulo que contém ao menos um componente que não possui dependências com nenhum outro componente neste mesmo módulo, o valor do custo total de comunicação recebe o valor de $CTC_{\max}$, indicando que a solução é indesejável, uma vez que é contrária à Premissa (1). Caso contrário, o custo total de comunicação consiste na soma do custo de comunicação introduzido por cada componente individualmente, de acordo com a função de *custo de comunicação* (CC), estando coerente com a Premissa (1).

Também conforme a Premissa (1), dado um componente c_i , o valor de CC_{c_i} é definido como sendo diretamente proporcional às dependências deste componente em relação aos demais componentes na DSM, tal como definido pela Equação (17). Nessa equação, n é o número de componentes na DSM, DSM_{c_j, c_i} é o valor da dependência do componente c_i em relação ao componente c_j de acordo com a DSM, e o termo W_{c_i, c_j} representa o peso da dependência entre os componentes c_i e c_j .

$$CC_{c_i} = \sum_{j=1}^n (DSM_{c_j, c_i} \times W_{c_i, c_j}) \quad (17)$$

O valor de W_{c_i, c_j} é expresso pela Equação (18), construída em conformidade com as Premissas (2) e (3). Nessa equação, t é o tamanho do módulo que contém os componentes c_i e c_j . Dado que um módulo é desenvolvido por uma única equipe e que dependências entre módulos sinalizam a necessidade de comunicação entre as equipes responsáveis por desenvolvê-los, a idéia da Equação (18) é demonstrar que componentes dependentes que são agrupados contribuem para a redução do valor de CC . Entretanto, se o tamanho dos módulos for elevado, a dependência interna também o será. De acordo com a Premissa (3), essa situação também influencia a necessidade de comunicação entre desenvolvedores, o que significa que a dependência interna ao módulo pode ser tão prejudicial quanto à existência de dependência externa. Dessa forma, quanto maior o módulo em termos da quantidade de componentes agrupados, maior é o peso da dependência interna ao módulo.

$$W_{c_i, c_j} = \begin{cases} \frac{t}{n}, & \text{se } c_i \text{ e } c_j \text{ estão no mesmo módulo} \\ 1, & \text{caso contrário} \end{cases} \quad (18)$$

5.2.3. Funcionamento Geral do Algoritmo

Dada a definição da função objetivo do algoritmo, nesta subseção é descrito o seu funcionamento geral, segundo o fluxograma apresentado pela Figura 8.

Primeiramente, no passo *inicializar*, o algoritmo atribui cada componente a um módulo. Com esta configuração, o valor inicial da função CTC é calculado, sendo o maior valor possível, já que nessa situação todas as dependências entre os componentes caracterizam dependências externas aos módulos.

Em seguida, o algoritmo seleciona um componente no passo *escolher componente randomicamente* e segue para o passo *mover componente randomicamente*, onde escolhe um módulo randomicamente e move o componente escolhido para ele, gerando uma nova solução de agrupamento. Então, no passo *calcular CTC*, o algoritmo calcula o valor da função CTC para essa nova solução e verifica se esse valor foi reduzido em relação à solução anterior no passo *o valor de CTC foi reduzido*.

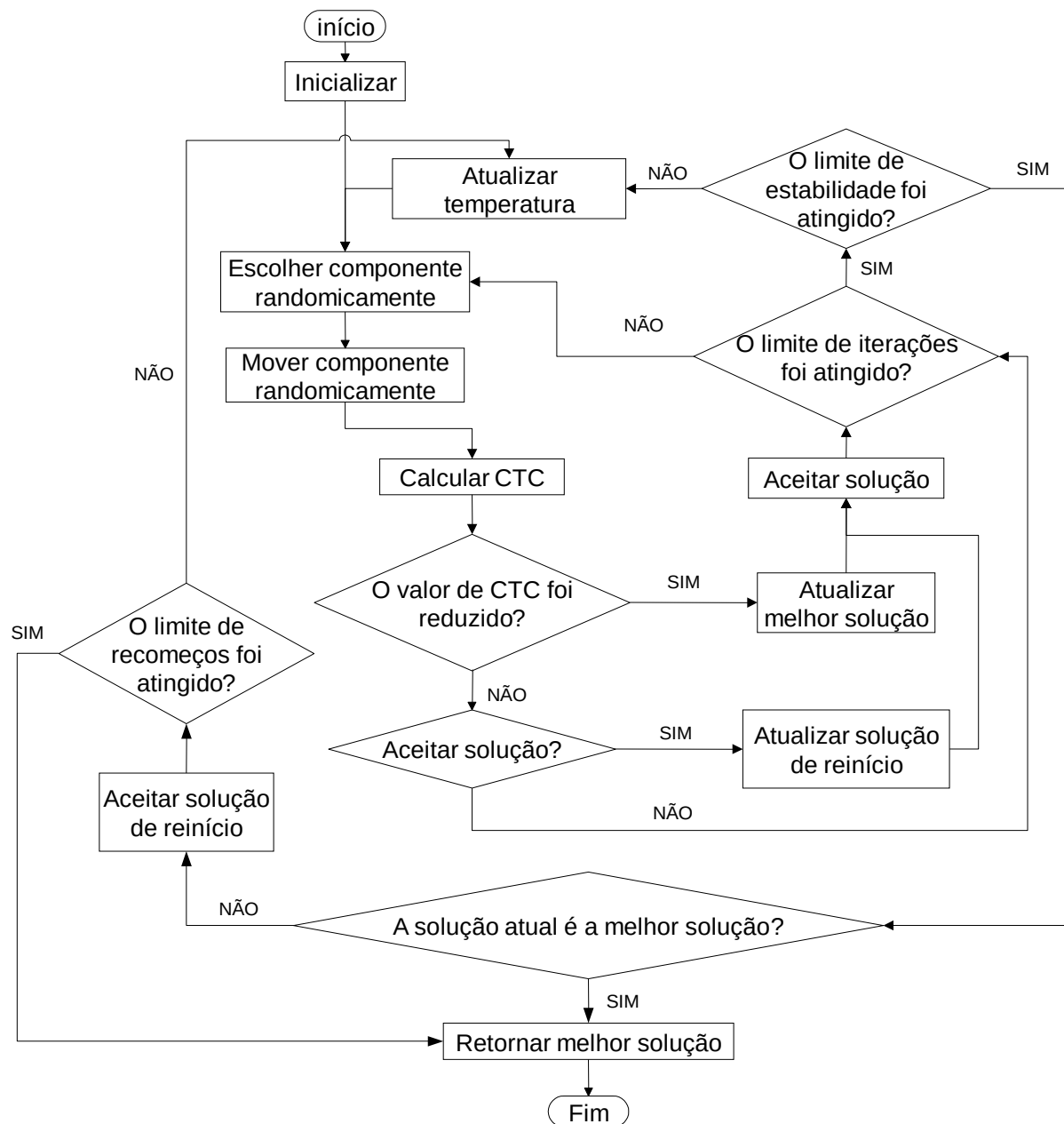


Figura 8. Fluxograma do algoritmo baseado em SA.

Em geral, o deslocamento do componente para o módulo se torna permanente quando promove a redução do valor de *CTC*. Quando isso acontece, o algoritmo salva a melhor solução encontrada até o momento, que é a solução corrente, no passo *atualizar melhor solução* e efetiva o deslocamento no passo *aceitar solução*. Contudo, para prevenir que o algoritmo fique preso a um ótimo local, é empregada a metaheurística *Simulated Annealing*, que faz o algoritmo explorar novas áreas no espaço de busca ao permitir que este aceite, com certa probabilidade, o deslocamento de um componente apesar de não ocorrer redução do valor de *CTC*. A decisão de aceitar uma solução de custo superior é tomada no passo *aceitar*

solução e depende do cálculo da *probabilidade de aceitação de uma solução de custo superior*.

A probabilidade de aceitação de uma solução de custo superior é expressa pela *equação de Metropolis*, em analogia ao processo da termodinâmica no qual SA está baseado. Essa equação foi introduzida em [Metropolis et al. 1953] e simula a possibilidade da movimentação de átomos que apresentam ganho de energia ser realizada numa determinada temperatura, conforme descrito a seguir pela Equação (19). Nessa equação, $P(\Delta E)$ é a probabilidade de movimentação do átomo, ΔE é a variação de energia causada por esta movimentação, k_B é a *constante de Boltzman* (dependente do material) e T é a temperatura do sistema.

$$P(\Delta E) = e^{-\Delta E / K_B T} \quad (19)$$

A metaheurística SA para problemas de otimização foi introduzida por [Kirkpatrick et al. 1983] utilizando a *equação de Metropolis* com $K_B = 1$, resultando na Equação (20), e desde então vem sendo utilizada dessa forma com excelentes resultados. Na Equação (20), T é um parâmetro de controle denominado *temperatura* e Δ consiste na diferença entre o custo total de comunicação da nova solução e da solução corrente, conforme especificado pela Equação (21).

$$P_{\text{aceitação}} = e^{(-\Delta / T)} \quad (20)$$

$$\Delta = CTC_{\text{nova}} - CTC_{\text{atual}} \quad (21)$$

O algoritmo gera um valor randômico x no intervalo fechado $[0, 1]$. Se $x < P_{\text{aceitação}}$, a solução que não promove a redução do custo total de comunicação é aceita. Dessa forma, o algoritmo segue para o passo *atualizar solução de reinício*, no qual a solução atual, a última solução aceita antes da nova solução, é memorizada, e em seguida, o algoritmo prossegue para o passo *aceitar solução*, a partir do qual a nova solução passa a ser a solução atual, significando que novas soluções serão geradas a partir dela.

Ao longo da busca, a temperatura T é resfriada de acordo com a Equação (22), que foi proposta por [Zolfaghari et al. 2002] e visa proporcionar um resfriamento mais rápido nas primeiras iterações e um resfriamento mais lento à medida que o número de iterações aumenta, minimizando buscas desnecessárias em temperaturas elevadas, quando a probabilidade de aceitação de uma solução de custo superior é elevada. Nessa equação, i é a

iteração corrente, T_i é a temperatura na iteração i , T_{i-1} é a temperatura na iteração anterior e T_0 é a temperatura inicial. O valor de T_0 é o valor inicial da função *CTC*.

$$T_i = \frac{(i - 1)T_{i-1} + T_0}{i(1 + \log i)} \quad (22)$$

Usando a mesma temperatura, o algoritmo repete o processo de gerar uma nova solução e decidir se esta deve ser aceita durante um número pré-definido de tentativas, especificado no parâmetro *limite de iterações* e avaliado no passo *o limite de estabilidade foi atingido*. A idéia de gerar várias soluções admitindo uma mesma temperatura é tentar alcançar a estabilidade da busca para a temperatura em questão. Dessa forma, esse valor deve ser elevado.

Após realizar todas as tentativas a uma mesma temperatura, no passo *o limite de estabilidade foi atingido*, o algoritmo checa se a busca atingiu a condição de estabilidade, especificada no parâmetro *limite de estabilidade*, que simplesmente é o número de vezes que a temperatura pode ser reduzida. Se o *limite de estabilidade* ainda não foi alcançado, o algoritmo atualiza a temperatura no passo *atualizar temperatura*, e procede para outro bloco de *limite de iterações* tentativas, como descrito anteriormente. Caso contrário, se o *limite de estabilidade* tiver sido alcançado, o algoritmo supõe que encontrou uma solução de agrupamento, e então, verifica se a solução atual é a melhor solução conhecida até o momento, tal como demonstrado pelo passo *a solução atual é a melhor solução*.

Se a solução atual é a melhor solução conhecida até o momento, a busca é finalizada e tal solução é retornada como indicado pelo passo *retornar melhor solução*. Caso contrário, a busca é reiniciada tomando como ponto de partida a *solução de reinício*, como indicado pelo passo *aceitar solução de reinício*. Observe que a *solução de reinício* é a solução anterior à última solução de custo superior que foi aceita pelo mecanismo de SA, e que foi previamente armazenada no passo *atualizar solução de reinício*, fazendo o algoritmo retroceder algumas soluções no espaço de busca e assim ter a chance de melhor explorá-lo.

Após recuperar a *solução de reinício*, no passo *o limite de recomeços foi atingido*, o algoritmo verifica se o número de reinícios foi alcançado, como denotado pelo parâmetro *limite de recomeços*. Se tal limite ainda não tiver sido alcançado, o algoritmo atualiza a temperatura no passo *atualizar temperatura*, e procede para outro bloco de *limite de iterações* tentativas, como já foi explicado. Caso contrário, a busca é finalizada e a melhor solução

encontrada é retornada. Os valores iniciais de *limite de iterações*, *limite de estabilidade* e *limite de recomenços* são definidos como parâmetros de entrada.

No caso de existir módulos previamente definidos, situação que ocorre quando um projeto é desenvolvido em múltiplas iterações, após a primeira iteração deve ser aplicada uma variação do algoritmo descrito, que recebe como entrada a DSM que contém todos os componentes, incluindo os que já foram agrupados em módulos, e a CMM que especifica tais módulos. A versão alternativa do algoritmo não permite que componentes previamente agrupados sejam movidos para outros módulos, embora permita que sejam introduzidos novos componentes em tais módulos. Nesse sentido, no passo *escolher componente randomicamente*, somente componentes que não foram previamente agrupados se tornam aptos para serem escolhidos.

Componentes em módulos previamente definidos já foram implementados. Dessa forma, movê-los para outro módulo causa a perda de informação sobre a equipe que os desenvolveram, informação esta que pode ser útil para fins de alocação. Quando novos componentes são introduzidos em módulos previamente definidos, a dependência deles com os componentes no módulo denota que a equipe que desenvolveu o módulo está apta para implementar esses novos componentes. E caso essa mesma equipe não esteja mais disponível e seja alocada uma nova equipe para o módulo reconfigurado, ao menos é possível prever a necessidade de comunicação que (provavelmente) existirá entre a equipe anterior e a nova equipe, o que irá influenciar na escolha da nova equipe.

5.3. Medição de Dependências entre Módulos

Uma vez que os módulos de componentes foram definidos, de acordo com a descrição da abordagem apresentada pela Figura 4, é preciso que seja fornecida informação sobre o nível de dependência entre módulos, a partir da qual é possível estimar a necessidade de comunicação entre as equipes que irão desenvolvê-los.

Na presente abordagem, essa informação é expressa através de uma matriz, denominada MBM (*Module Based Matrix*). Trata-se de uma matriz semelhante à DSM, que identifica a dependência na perspectiva de módulos ao invés de componentes. Essa matriz é construída com base na CMM gerada como saída do algoritmo de agrupamento e na DSM do projeto. A diagonal principal da MBM identifica a dependência interna de cada módulo, enquanto que as demais células identificam a dependência entre módulos.

Considerando os módulos $M_A = \{c_{A1}, c_{A2}, \dots, c_{Ax}\}$ e $M_B = \{c_{B1}, c_{B2}, \dots, c_{By}\}$, de acordo com a Equação (23), a dependência do módulo M_A em relação ao módulo M_B ($MBM'_{MB, MA}$) consiste no somatório da dependência de cada componente de M_A em relação a cada componente de M_B . Igualmente, a dependência interna ao módulo M_A ($MBM'_{MA, MA}$) consiste no somatório da dependência entre cada componente de M_A e os demais componentes neste módulo.

$$MBM'_{MB, MA} = \sum_{i=1}^x \sum_{j=1}^y DSM_{c_{Bj}, c_{Ai}} \quad (23)$$

Para manter o resultado desta métrica no intervalo fechado $[0,1]$, em concordância com as métricas anteriormente apresentadas, é aplicada a fórmula de normalização descrita pela Equação (24). Nessa equação, MBM_{max} é o maior valor identificado na MBM antes da normalização.

$$MBM_{MB, MA} = \frac{MBM'_{MB, MA}}{MBM_{max}} \quad (24)$$

5.4. Considerações Finais

Neste capítulo, foi apresentada a abordagem desenvolvida para recomendar módulos em projetos distribuídos de LPS. Trata-se de uma abordagem flexível, organizada em três etapas com entrada e saída bem definidas, que funcionam de forma relativamente independente, passíveis de serem adaptadas ou estendidas sem requerer modificações nas demais.

A etapa de *Medição de dependências entre componentes* define medidas quantitativas para caracterizar dependências em nível arquitetural entre componentes segundo as interfaces e operações providas e requeridas destes, e as *features* por eles envolvidas. A definição de interfaces/operações providas e requeridas é própria do desenvolvimento baseado em componentes, e reflete o acoplamento entre componentes, ou seja, o quanto um componente precisa de outro. Por sua vez, a modelagem de *features* é típica de projetos de LPS e reflete a coesão dos componentes, mostrando o quanto eles estão relacionados em termos de funcionalidade. Dado que a dependência entre componentes é medida em termos de acoplamento e coesão, o uso combinado dessas métricas é recomendado, apesar de também poderem ser usadas individualmente.

Já a etapa de *Recomendação de módulos* define um algoritmo de agrupamento baseado em SA para recomendar módulos fracamente dependentes entre si, constituídos por

componentes fortemente dependentes. A metaheurística SA permite que o algoritmo explore melhor o espaço de busca do problema, aumentando as chances de encontrar a solução ótima ou uma solução próxima da ótima. Apesar disso, é recomendado que o algoritmo seja executado mais de uma vez para uma mesma iteração de desenvolvimento, já que por envolver aspectos randômicos, ele poderá resultar em soluções diferentes para uma mesma entrada.

Por fim, a etapa de *Medição de dependências entre módulos* define uma medida quantitativa para caracterizar dependências entre módulos. Essa medida auxilia o gerente de projeto na tomada de decisão sobre como alocar equipes aos módulos, uma vez que a partir do nível de dependência entre módulos é possível inferir sobre a necessidade de comunicação entre as equipes que irão desenvolvê-los. O gerente deve procurar minimizar a dificuldade de comunicação entre equipes que devem interagir mais intensamente. Dessa forma, características das equipes como localização, fuso-horário, horário de trabalho e conhecimento sobre idiomas precisam ser consideradas, além da capacidade técnica das equipes para implementar os módulos, que é o pré-requisito para uma equipe poder ser alocada a um módulo.

Capítulo 6

Estudos de Caso

Neste capítulo são apresentados quatro estudos de caso realizados com o propósito de avaliar a abordagem desenvolvida, além de demonstrar o seu uso. Primeiramente são apresentados dois estudos de caso de linhas de produto hipotéticas, documentadas em [Gomaa 2004]: uma linha de produto de microondas, a partir da qual foi feita a avaliação das métricas DI e DO; e uma linha de produto de comércio eletrônico (E-Commerce), que foi organizada em duas iterações de desenvolvimento para avaliar o uso da abordagem nessa situação. Em seguida, são apresentados dois estudos de caso de projetos reais a partir dos quais foi possível fazer uma avaliação comparativa entre as soluções encontradas pelo algoritmo com soluções definidas pelo arquiteto do projeto: Multi-MOM [Morais, Elias 2010], uma linha de produtos de *middleware* para computação móvel; e Ginga-CC [Souza Filho et al. 2007], uma linha de produtos de *middleware* para aplicações de TV digital.

Para todos esses estudos de caso, os componentes nos diagramas arquiteturais estão enumerados para que possam ser identificados nas matrizes, e pontos de variação são representados por um componente genérico que contém mais de um índice, cada um deles correspondendo a um componente variante. Essa representação genérica abstrai as diferenças entre os componentes variantes, sendo adotada apenas para prover uma representação única da arquitetura. Entretanto, o cálculo da métrica DI deve ser realizado com base na especificação real de cada componente e por isso, quando necessário, tais componentes são detalhados. Já a informação sobre as *features* implementadas pelos componentes, bem como suas operações providas e requeridas, são resumidas em tabelas.

Os parâmetros do algoritmo de agrupamento (*iterações por temperatura*, *limite de estabilidade* e *limite de recomenços*) foram definidos de acordo com o tamanho do espaço de busca em questão. Inicialmente, inclusive, foi tentado definir uma forma genérica de calcular tais parâmetros automaticamente, como por exemplo, definir uma porcentagem do total de soluções para serem investigadas. Entretanto, dependendo do problema, uma porcentagem

reduzida pode representar uma quantidade de soluções tão elevada, que se torne impraticável. Esse é o caso da segunda iteração da linha de produtos de E-commerce (26 componentes).

Dessa forma, a definição dos parâmetros foi feita de maneira experimental, tomando por base a quantidade de componentes e não o total de soluções de agrupamento possíveis, mantendo a seguinte proporção:

- $Iterações = componentes \times 20$
- $Estabilidade = iterações + 100$
- $Recomeços = componentes/3$

A Tabela 6 descreve a configuração dos parâmetros para cada estudo de caso.

Tabela 6. Valores adotados para os parâmetros do algoritmo baseado em SA.

Estudo de caso	Quantidade de componentes	Total de soluções	Iterações por temperatura	Limite de estabilidade	Limite de Recomeços
Microondas	21	474.869.816.156.751	420	520	7
E-commerce (iteração 1)	12	4.213.597	240	340	4
E-commerce (iteração 2)	26	49.631.246.523.618.700.000	520	620	9
Multi-MOM	12	4.213.597	240	340	4
Ginga-CC	12	4.213.597	240	340	4

6.1. Linha de Produto de Microondas

A arquitetura da linha de produtos de microondas é apresentada na Figura 9. Nessa figura podem ser identificados componentes opcionais (estereótipo <<optional>>), obrigatórios (estereótipos <<kernel>> e <<kernel-param-vp>>) e variáveis (estereótipo <<variant>>). O componente *Microwave Control* é um componente composto e para os propósitos da presente abordagem são considerados os seus componentes constituintes *Microwave Oven Control* (c_4), *Oven Timer* (c_{17}), *Oven Data* (c_{18}), *TOD Timer* (c_{19}) e *Recipes* (c_{20}), explicitados na figura.

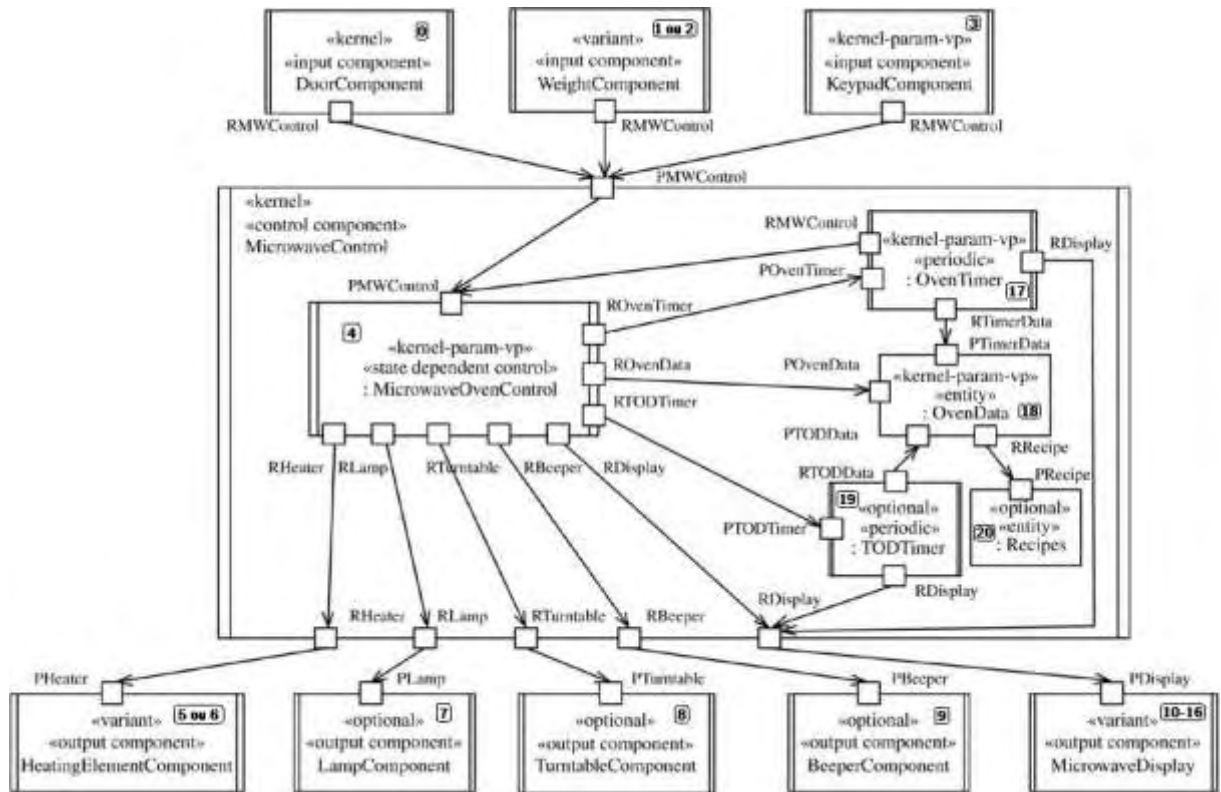


Figura 9. Arquitetura da LPS de microondas [Gomaa 2004].

O componente *Microwave Display* também é um componente composto, que compreende dois pontos de variação, os quais são genericamente representados na Figura 10 pelos componentes *Display Interface* (*DisplayInterface-OneLine* e *DisplayInterface-MultiLine*) e *Display Prompt* (*DisplayPrompt-English*, *DisplayPrompt-French*, *DisplayPrompt-Spanish*, *DisplayPrompt-German* e *DisplayPrompt-Italian*). Observe que apesar de existirem 2 (duas) variantes para o componente *Display Interface* e 5 (cinco) variantes para o componente *Display Prompt*, uma aplicação deve possuir apenas um componente de cada tipo. Dessa forma, o cálculo das métricas DI e DO deve considerar o número real de interfaces/operações requeridas pelos componentes, e não contabilizar cada variante individualmente.

Por exemplo, o cálculo correto da dependência do componente c_{10} (*DisplayInterface-OneLine*) em relação ao componente c_{12} (*DisplayPrompt-English*) é $DI_{c_{10}, c_{12}} = Pc_{12}, c_{10} / Rc_{10} = 1/1 = 1$, e não $DI_{c_{10}, c_{12}} = 1/5 = 0,2$, porque c_{10} requer apenas uma interface e não cinco (caso que consideraria uma interface de cada componente variante do tipo *Display Prompt*).

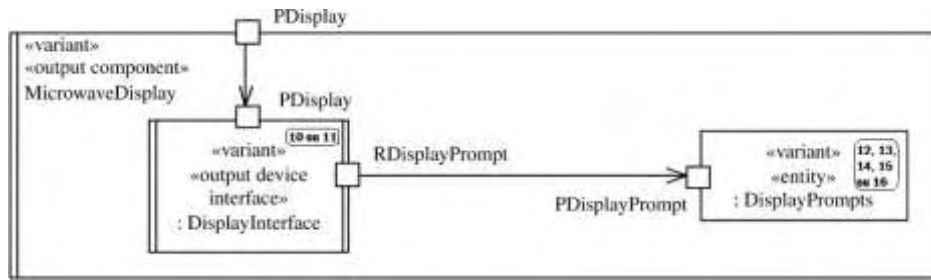
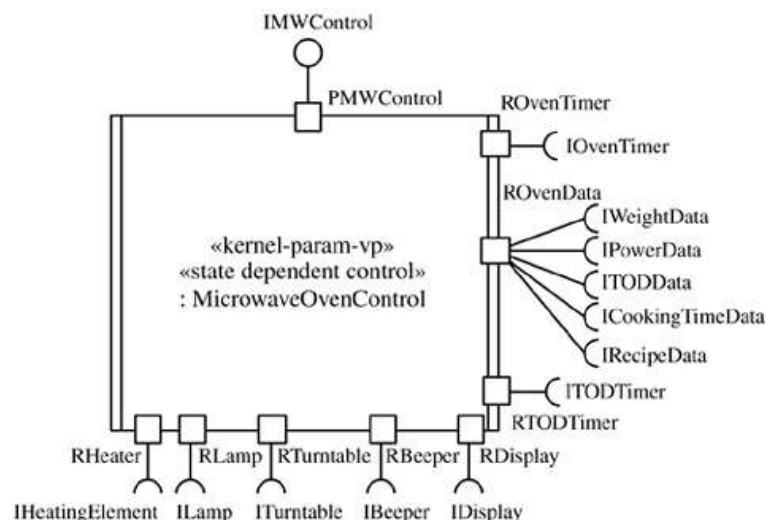


Figura 10. Componente *Microwave Display* [Gomaa 2004].

Como em todos os pontos de variação (componentes genéricos *Weight Component*, *Heating Element Component*, *Display Interface* e *Display Prompts*) os componentes variantes apresentam a mesma especificação, não se fez necessário detalhá-los aqui. Já os componentes *Keypad*, *Microwave Oven Control*, *Oven Timer* e *Oven Data* são parametrizáveis, o que significa que possuem implementação única, mas podem variar de comportamento conforme os parâmetros passados, os quais podem ser configurados em tempo de compilação ou de execução. Nesta linha de produtos, os parâmetros consistem nas *features* envolvidas pelos componentes. O componente *Keypad* requer uma única interface (*IMWControl*) e envolve as *features* *Power Level*, *Recipe*, *TOD Clock* e *Minute Plus*. De acordo com a *feature* selecionada, a partir desse componente podem ser disparadas diferentes ações, conforme o parâmetro passado para a única operação de sua interface requerida.

Os componentes *Microwave Oven Control*, *Oven Timer* e *Oven Data* são detalhados na Figura 11, já que na descrição arquitetural a definição de portas abstrai suas interfaces providas e requeridas. Por meio de tais componentes é possível habilitar e desabilitar interfaces providas e/ou requeridas, bem como habilitar e desabilitar operações nas interfaces, segundo as *features* selecionadas.



(a) Componente *Microwave Oven Control*.

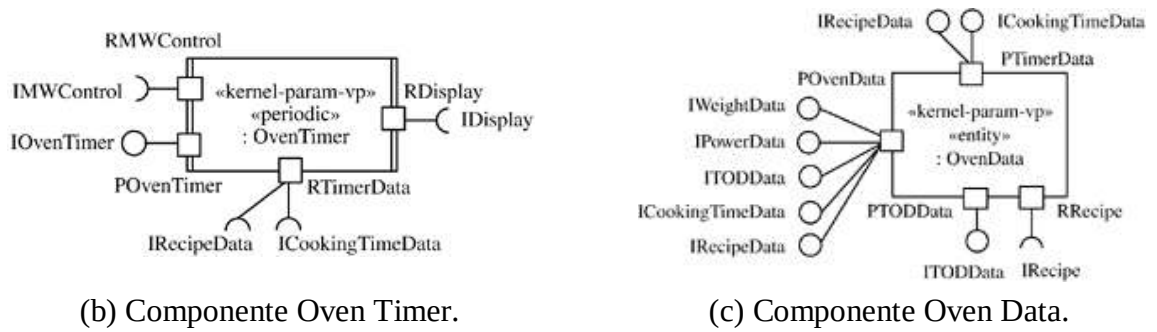


Figura 11. Visão detalhada dos componentes parametrizáveis [Gomaa 2004].

A Tabela 7 apresenta as *features* implementadas por cada componente, enquanto que a Figura 12 apresenta a DSM resultante do uso da métrica DF, cujos valores diferentes de zero foram sombreados para facilitar a visualização.

Tabela 7. Relação de *features* por componente na LPS de microondas.

Índice	Componente	Feature	Quantidade
c_0	Door	MicrowaveOvenKernel	1
c_1	WeightSensor-Analog	MicrowaveOvenKernel	2
		AnalogWeight	
c_2	WeightSensor-Boolean	MicrowaveOvenKernel	2
		BooleanWeight	
c_3	Keypad	MicrowaveOvenKernel	5
		MinutePlus	
		PowerLevel	
		TODClock	
		Recipe	
c_4	MicrowaveOvenControl	MicrowaveOvenKernel	9
		Light	
		Turntable	
		Beeper	
		MinutePlus	
		MultiLevelHeating	
		PowerLevel	
		TODClock	
		Recipe	
c_5	HeatingElement-OneLevel	MicrowaveOvenKernel	2
		OneLevelHeating	
c_6	HeatingElement-	MicrowaveOvenKernel	2

	MultiLevel	MultiLevelHeating	
c_7	Lamp	Light	1
c_8	Turntable	Turntable	1
c_9	Beeper	Beeper	1
c_{10}	DisplayInterface-OneLine	MicrowaveOvenKernel	2
		OneLineDisplay	
c_{11}	DisplayInterface-MultiLine	MicrowaveOvenKernel	2
		MultiLineDisplay	
c_{12}	DisplayPrompt-English	MicrowaveOvenKernel	2
		English	
c_{13}	DisplayPrompt-French	MicrowaveOvenKernel	2
		French	
c_{14}	DisplayPrompt-Spanish	MicrowaveOvenKernel	2
		Spanish	
c_{15}	DisplayPrompt-German	MicrowaveOvenKernel	2
		German	
c_{16}	DisplayPrompt-Italian	MicrowaveOvenKernel	2
		Italian	
c_{17}	OvenTimer	MicrowaveOvenKernel	3
		MinutePlus	
		Recipe	
c_{18}	OvenData	MicrowaveOvenKernel	7
		MinutePlus	
		AnalogWeight	
		MultiLevelHeating	
		TODClock	
		12/24 HourClock	
		Recipe	
c_{19}	TODTimer	TODClock	1
c_{20}	Recipes	Recipe	1

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
c0	0,00	0,50	0,50	0,20	0,11	0,50	0,50	0,00	0,00	0,00	0,50	0,50	0,50	0,50	0,50	0,50	0,50	0,33	0,14	0,00	0,00
c1	0,50	0,00	0,33	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,33	0,33	0,25	0,29	0,00	0,00
c2	0,50	0,33	0,00	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,33	0,33	0,25	0,13	0,00	0,00
c3	0,20	0,17	0,17	0,00	0,56	0,17	0,17	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,60	0,71	0,20	0,20
c4	0,11	0,10	0,10	0,56	0,00	0,10	0,22	0,11	0,11	0,11	0,10	0,10	0,10	0,10	0,10	0,10	0,10	0,33	0,45	0,11	0,11
c5	0,50	0,33	0,33	0,17	0,10	0,00	0,33	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,33	0,33	0,25	0,13	0,00	0,00
c6	0,50	0,33	0,33	0,17	0,22	0,33	0,00	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,33	0,33	0,25	0,29	0,00	0,00
c7	0,00	0,00	0,00	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c8	0,00	0,00	0,00	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c9	0,00	0,00	0,00	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c10	0,50	0,33	0,33	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,33	0,25	0,13	0,00	0,00
c11	0,50	0,33	0,33	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,33	0,00	0,33	0,33	0,33	0,33	0,33	0,25	0,13	0,00	0,00
c12	0,50	0,33	0,33	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,33	0,33	0,00	0,33	0,33	0,33	0,33	0,25	0,13	0,00	0,00
c13	0,50	0,33	0,33	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,33	0,33	0,33	0,00	0,33	0,33	0,33	0,25	0,13	0,00	0,00
c14	0,50	0,33	0,33	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,00	0,33	0,33	0,25	0,13	0,00	0,00
c15	0,50	0,33	0,33	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,00	0,33	0,25	0,13	0,00	0,00
c16	0,50	0,33	0,33	0,17	0,10	0,33	0,33	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,33	0,00	0,25	0,13	0,00	0,00
c17	0,33	0,25	0,25	0,60	0,33	0,25	0,25	0,00	0,00	0,00	0,25	0,25	0,25	0,25	0,25	0,25	0,25	0,00	0,43	0,00	0,00
c18	0,14	0,29	0,13	0,71	0,45	0,13	0,29	0,00	0,00	0,00	0,13	0,13	0,13	0,13	0,13	0,13	0,13	0,43	0,00	0,14	0,14
c19	0,00	0,00	0,00	0,20	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,14	0,00	0,00
c20	0,00	0,00	0,00	0,20	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,14	0,00	0,00

Figura 12. DSM da LPS de microondas segundo a métrica DF.

O estudo de caso dessa linha de produtos considera que todos os componentes são implementados em uma única iteração de desenvolvimento, de duas maneiras: utilizando-se as métricas DI e DF e utilizando-se as métricas DO e DF. O objetivo é comparar o uso das métricas DI e DO. As subseções 6.1.1 e 6.2.2 a seguir detalham os resultados obtidos utilizando-se cada um desses conjuntos de métricas.

6.1.1. Caso Baseado nas Métricas DI e DF

A DSM da linha de produto de microondas segundo a métrica DI é descrita pela Figura 13, na qual os valores diferentes de zero foram sombreados para facilitar a visualização. De acordo com o cálculo da métrica DF, discriminado pela DSM da Figura 12, e pelo cálculo da métrica DI, discriminado pela DSM da Figura 13, foi usada a Equação (12) com $\alpha = 2/3$ para combinar as métricas DI e DF, resultando na construção da DSM da Figura 14. A fim de demonstrar como os valores foram definidos nessa matriz, considere os componentes *Microwave Oven Control* (c_4) e *Oven Data* (c_{18}). c_4 requer 12 interfaces (Figura 11a) e c_{18} provê 5 delas (Figura 11c). Então, $DI_{c_4, c_{18}} = 5/12 \cong 0,417$. Além disso, c_4 e c_{18} implementam 5 *features* em comum e totalizam 11 *features* (Tabela 7), o que significa que $DF_{c_4, c_{18}} = \frac{5}{11} \cong 0,45$. Como resultado, a dependência do componente c_4 para o componente c_{18} é $DSM_{c_{18}, c_4} = [0,417 \times (2/3)] + [0,45 \times (1/3)] \cong 0,43$.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
c0	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c4	1,00	1,00	1,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,25	0,00	0,00	0,00
c5	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c6	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c7	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c8	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c9	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c10	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,25	0,00	0,50	0,00
c11	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,25	0,00	0,50	0,00
c12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c13	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c14	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c15	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c16	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c17	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c18	0,00	0,00	0,00	0,00	0,42	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,50	0,00	0,50	0,00
c19	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c20	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00	0,00	0,00

Figura 13. DSM da LPS de microondas segundo a métrica DI.

CTC ≈ 36,9																					
	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
c0	0,00	0,17	0,17	0,07	0,04	0,17	0,17	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,05	0,00	0,00
c1	0,17	0,00	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,08	0,10	0,00	0,00
c2	0,17	0,11	0,00	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,08	0,04	0,00	0,00
c3	0,07	0,06	0,06	0,00	0,19	0,06	0,06	0,00	0,00	0,00	0,06	0,06	0,06	0,06	0,06	0,06	0,06	0,20	0,24	0,07	0,07
c4	0,70	0,70	0,70	0,85	0,00	0,03	0,07	0,04	0,04	0,04	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,28	0,15	0,04	0,04
c5	0,17	0,11	0,11	0,06	0,09	0,00	0,11	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,08	0,04	0,00	0,00
c6	0,17	0,11	0,11	0,06	0,13	0,11	0,00	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,08	0,10	0,00	0,00
c7	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c8	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c9	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c10	0,17	0,11	0,11	0,06	0,09	0,11	0,11	0,00	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,25	0,04	0,33	0,00
c11	0,17	0,11	0,11	0,06	0,09	0,11	0,11	0,00	0,00	0,00	0,11	0,00	0,11	0,11	0,11	0,11	0,11	0,25	0,04	0,33	0,00
c12	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,00	0,11	0,11	0,11	0,11	0,08	0,04	0,00	0,00
c13	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,11	0,00	0,11	0,11	0,11	0,08	0,04	0,00	0,00
c14	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,11	0,11	0,00	0,11	0,11	0,08	0,04	0,00	0,00
c15	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,11	0,11	0,11	0,00	0,11	0,08	0,04	0,00	0,00
c16	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,11	0,11	0,11	0,11	0,00	0,08	0,04	0,00	0,00
c17	0,11	0,08	0,08	0,20	0,17	0,08	0,08	0,00	0,00	0,00	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,00	0,14	0,00	0,00
c18	0,05	0,10	0,04	0,24	0,43	0,04	0,10	0,00	0,00	0,00	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,48	0,00	0,38	0,05
c19	0,00	0,00	0,00	0,07	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,05	0,00	0,00
c20	0,00	0,00	0,00	0,07	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,71	0,00	0,00

Figura 14. DSM da LPS de microondas segundo as métricas DI e DF.

A solução de agrupamento encontrada pelo algoritmo é descrita pela CMM da Figura 15 e a DSM reordenada da Figura 16. Conforme pode ser observado, os componentes compostos foram preservados, ou seja, seus componentes constituintes foram agrupados juntos: os componentes genéricos *Display Interface* e *Display Prompt* foram agrupados, em conformidade com o componente composto *Microwave Display*; assim como os componentes *Microwave Oven Control* (c_4), *Oven Timer* (c_{17}), *Oven Data* (c_{18}), *TOD Timer* (c_{19}) e *Recipes*

(c_{20}) foram agrupados juntos, em concordância com o componente composto *Microwave Control*.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
M0	1	0	0	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0	0	0
M1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
M2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
M3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
M4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Figura 15. CMM da solução encontrada para a LPS de microondas.

CTC ≈ 22,83																					
	c0	c5	c6	c10	c11	c12	c13	c14	c15	c16	c1	c2	c3	c4	c17	c18	c19	c20	c7	c9	c8
M0	c0	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,07	0,04	0,11	0,05	0,00	0,00	0,00	0,00	0,00
	c5	0,17	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,06	0,09	0,08	0,04	0,00	0,00	0,00	0,00	0,00
	c6	0,17	0,11	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,06	0,13	0,08	0,10	0,00	0,00	0,00	0,00	0,00
	c10	0,17	0,11	0,11	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,06	0,09	0,25	0,04	0,33	0,00	0,00	0,00	0,00
	c11	0,17	0,11	0,11	0,11	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,06	0,09	0,25	0,04	0,33	0,00	0,00	0,00	0,00
	c12	0,17	0,11	0,11	0,78	0,78	0,00	0,11	0,11	0,11	0,11	0,11	0,06	0,03	0,08	0,04	0,00	0,00	0,00	0,00	0,00
	c13	0,17	0,11	0,11	0,78	0,78	0,11	0,00	0,11	0,11	0,11	0,11	0,06	0,03	0,08	0,04	0,00	0,00	0,00	0,00	0,00
	c14	0,17	0,11	0,11	0,78	0,78	0,11	0,11	0,00	0,11	0,11	0,11	0,06	0,03	0,08	0,04	0,00	0,00	0,00	0,00	0,00
M1	c15	0,17	0,11	0,11	0,78	0,78	0,11	0,11	0,11	0,00	0,11	0,11	0,06	0,03	0,08	0,04	0,00	0,00	0,00	0,00	0,00
	c16	0,17	0,11	0,11	0,78	0,78	0,11	0,11	0,11	0,00	0,11	0,11	0,06	0,03	0,08	0,04	0,00	0,00	0,00	0,00	0,00
	c1	0,17	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,00	0,11	0,06	0,03	0,08	0,10	0,00	0,00	0,00	0,00	0,00
	c2	0,17	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,00	0,06	0,03	0,08	0,04	0,00	0,00	0,00	0,00	0,00
	c3	0,07	0,06	0,06	0,06	0,06	0,06	0,06	0,06	0,06	0,06	0,06	0,00	0,19	0,20	0,24	0,07	0,07	0,00	0,00	0,00
	c4	0,70	0,03	0,07	0,03	0,03	0,03	0,03	0,03	0,03	0,70	0,70	0,85	0,00	0,28	0,15	0,04	0,04	0,04	0,04	0,04
	c17	0,11	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,20	0,17	0,00	0,14	0,00	0,00	0,00	0,00	0,00
	c18	0,05	0,04	0,10	0,04	0,04	0,04	0,04	0,04	0,04	0,10	0,04	0,24	0,43	0,48	0,00	0,38	0,05	0,00	0,00	0,00
M2	c19	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,07	0,09	0,00	0,05	0,00	0,00	0,00	0,00	0,00
	c20	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,07	0,04	0,00	0,71	0,00	0,00	0,00	0,00	0,00
	c7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	c9	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M4	c8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Figura 16. DSM da LPS de microondas reordenada.

Os componentes *Door* (c_0), *Heating Element – One Level* (c_5) e *Heating Element – Multi Level* (c_6) ainda foram agrupados com o componente composto *Microwave Display* porque, apesar desses componentes não manterem relacionamento direto através de interfaces e operações providas e requeridas, apresentam *features* em comum. Isso pode ser verificado na DSM da Figura 12, segundo a qual $DF_{c_0, c_5} = DF_{c_5, c_0} = 0,5$, $DF_{c_0, c_6} = DF_{c_6, c_0} = 0,5$ e $DF_{c_5, c_6} = DF_{c_6, c_5} = 0,33$.

Já os componentes *Weight Sensor – Analog* (c_1), *Weight Sensor – Boolean* (c_2) e *Keypad Component* (c_3) também foram agrupados com o componente composto *Microwave Control* devido ao elevado nível de dependência desses componentes em relação ao componente *Microwave Oven Control* (c_4), que é resultante do relacionamento entre eles através de interfaces providas e requeridas, conforme pode ser verificado na DSM da Figura 13, segundo a qual $DIC_{c_1, c_4} = DIC_{c_2, c_4} = DIC_{c_3, c_4} = 1,0$.

Por fim, os componentes *Lamp* (c_7), *Turntable* (c_8) e *Beeper* (c_9) ficaram em módulos isolados, porque só possuem dependência em relação ao componente *Microwave Oven Control* (c_4), e mesmo assim, em um nível tão baixo que não compensou o aumento do tamanho do módulo que envolve o componente c_4 . De acordo com a DSM composta da Figura 14, $DSM_{c_7, c_4} = DSM_{c_8, c_4} = DSM_{c_9, c_4} = 0,09$ e $DSM_{c_4, c_7} = DSM_{c_4, c_8} = DSM_{c_4, c_9} = 0,04$.

Uma solução de agrupamento que poderia ser definida de maneira intuitiva, apenas com base no diagrama arquitetural, é descrita pela CMM da Figura 17. Entretanto, essa solução despreza os relacionamentos entre os componentes definidos a partir das *features* em comum por eles envolvidas, os quais influenciam no relacionamento entre os componentes e consequentemente entre as equipes responsáveis por implementá-los. Dessa forma, o custo da solução alternativa é superior ($\cong 26,63$) ao da solução encontrada pelo algoritmo ($\cong 22,83$). Esse fenômeno é demonstrado pela DSM descrita pela Figura 18, que foi reordenada de acordo com a CMM da Figura 17. Na Figura 18, os valores em células tracejadas estão integrados ao módulo M_0 na solução encontrada pelo algoritmo (Figura 16), enquanto que os valores em negrito em células pontilhadas estão adicionados ao módulo M_1 .

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
M0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0
M1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
M2	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M3	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
M7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
M8	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Figura 17. CMM da solução alternativa para a LPS de microondas.

Ademais, como o projeto contém 21 componentes, o que resulta num espaço de busca com 474.869.816.156.751 soluções, não foi possível avaliar todas as soluções possíveis para verificar se a solução encontrada pelo algoritmo é a solução ótima.

		CTC ≈ 26,63																				
		c10	c11	c12	c13	c14	c15	c16	c4	c17	c18	c19	c20	c1	c2	c5	c6	c0	c3	c7	c8	c9
M0	c10	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,09	0,25	0,04	0,33	0,00	0,11	0,11	0,11	0,11	0,17	0,06	0,00	0,00	0,00
	c11	0,11	0,00	0,11	0,11	0,11	0,11	0,11	0,09	0,25	0,04	0,33	0,00	0,11	0,11	0,11	0,11	0,17	0,06	0,00	0,00	0,00
	c12	0,78	0,78	0,00	0,11	0,11	0,11	0,11	0,03	0,08	0,04	0,00	0,00	0,11	0,11	0,11	0,11	0,17	0,06	0,00	0,00	0,00
	c13	0,78	0,78	0,11	0,00	0,11	0,11	0,11	0,03	0,08	0,04	0,00	0,00	0,11	0,11	0,11	0,11	0,17	0,06	0,00	0,00	0,00
	c14	0,78	0,78	0,11	0,11	0,00	0,11	0,11	0,03	0,08	0,04	0,00	0,00	0,11	0,11	0,11	0,11	0,17	0,06	0,00	0,00	0,00
	c15	0,78	0,78	0,11	0,11	0,11	0,00	0,11	0,03	0,08	0,04	0,00	0,00	0,11	0,11	0,11	0,11	0,17	0,06	0,00	0,00	0,00
M1	c16	0,78	0,78	0,11	0,11	0,11	0,11	0,00	0,03	0,08	0,04	0,00	0,00	0,11	0,11	0,11	0,11	0,17	0,06	0,00	0,00	0,00
	c4	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,00	0,28	0,15	0,04	0,04	0,70	0,70	0,03	0,07	0,70	0,85	0,04	0,04	0,04
	c17	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,17	0,00	0,14	0,00	0,00	0,08	0,08	0,08	0,08	0,11	0,20	0,00	0,00	0,00
	c18	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,43	0,48	0,00	0,38	0,05	0,10	0,04	0,04	0,10	0,05	0,24	0,00	0,00	0,00
	c19	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,09	0,00	0,05	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,07	0,00	0,00	0,00
M2	c20	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,04	0,00	0,71	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,07	0,00	0,00	0,00
	c1	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,03	0,08	0,10	0,00	0,00	0,00	0,11	0,11	0,11	0,17	0,06	0,00	0,00	0,00
M3	c2	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,03	0,08	0,04	0,00	0,00	0,11	0,00	0,11	0,11	0,17	0,06	0,00	0,00	0,00
	c5	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,09	0,08	0,04	0,00	0,00	0,11	0,11	0,00	0,11	0,17	0,06	0,00	0,00	0,00
	c6	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,13	0,08	0,10	0,00	0,00	0,11	0,11	0,11	0,00	0,17	0,06	0,00	0,00	0,00
M4	c0	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,04	0,11	0,05	0,00	0,00	0,17	0,17	0,17	0,17	0,00	0,07	0,00	0,00	0,00
M5	c3	0,06	0,06	0,06	0,06	0,06	0,06	0,06	0,19	0,20	0,24	0,07	0,07	0,06	0,06	0,06	0,06	0,07	0,00	0,00	0,00	0,00
M6	c7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M7	c8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M8	c9	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Figura 18. DSM da LPS de microondas reordenada de acordo com a solução alternativa.

A MBM associada à solução encontrada pelo algoritmo é descrita pela Figura 19. Nessa figura, as células sombreadas da matriz representam a dependência interna aos módulos, enquanto que os demais valores representam as dependências entre módulos. Como em todos os módulos há componentes relacionados ao componente *Microwave Oven Control* (c_4), pertencente ao módulo M_1 , todos os demais módulos se relacionam ao módulo M_1 , não sendo possível minimizar eficientemente as dependências externas a este módulo. Apesar disso, o somatório das dependências internas aos módulos (1,43948) é superior ao somatório das dependências externas entre eles (0,64762), o que mostra que a solução está coerente com os propósitos definidos pela abordagem.

	M_0	M_1	M_2	M_3	M_4
M_0	1,00000	0,32919	0,00000	0,00000	0,00000
M_1	0,29638	0,43948	0,00226	0,00226	0,00226
M_2	0,00000	0,00509	0,00000	0,00000	0,00000
M_3	0,00000	0,00509	0,00000	0,00000	0,00000
M_4	0,00000	0,00509	0,00000	0,00000	0,00000

Figura 19. MBM da LPS de microondas segundo as métricas DI e DF.

6.1.2. Caso Baseado nas Métricas DO e DF

A fim de identificar o relacionamento de dependência entre componentes segundo a métrica DO, fez-se necessário identificar as operações providas e requeridas pelos componentes, segundo suas interfaces providas e requeridas, respectivamente. A Tabela 8 identifica as interfaces requeridas por cada componente.

Tabela 8. Interfaces requeridas pelos componentes da LPS de microondas.

Índice	Componente	Interface Requerida
c_0	Door	IMWControl
c_1	WeightSensor -Analog	IMWControl
c_2	WeightSensor-Boolean	IMWControl
c_3	Keypad	IMWControl
c_4	MicrowaveOvenControl	IHeatingElement
		ILamp
		ITurntable
		IBeeper
		IDisplay
		ITODTimer
		IRecipeData
		ICookingTimeData
		ITODData
		IPowerData
		IWeightData
		IOvenTimer
c_5	HeatingElement-OneLevel	-
c_6	HeatingElement-MultiLevel	-
c_7	Lamp	-
c_8	Turntable	-
c_9	Beeper	-
c_{10}	DisplayInterface-OneLine	IDisplayPrompt
c_{11}	DisplayInterface-MultiLine	IDisplayPrompt
c_{12}	DisplayPrompt-English	-
c_{13}	DisplayPrompt-French	-
c_{14}	DisplayPrompt-Spanish	-
c_{15}	DisplayPrompt-German	-
c_{16}	DisplayPrompt-Italian	-
c_{17}	OvenTimer	IMWControl
		IDisplay
		IRecipeData

		ICookingTimeData
c₁₈	OvenData	IRecipe
c₁₉	TOTimer	ITODData
		IDisplay
c₂₀	Recipes	-

Já a Tabela 9 identifica as operações providas por cada interface. Nessa tabela, a interface *ILamp*, provida pelo componente opcional *Lamp* (*c₇*), só pode ser usada se a *feature Light* for escolhida. O mesmo se aplica para as demais interfaces e operações em que há *features* especificadas. Nesse contexto é válido reforçar que, apesar do relacionamento entre os componentes só efetivamente existirem quando estes são escolhidos e introduzidos numa arquitetura, quando um dado componente é incluído em uma iteração de desenvolvimento, é relevante para o presente trabalho extrair informações sobre todos os relacionamentos que o componente pode suportar. Nesse sentido, mesmo que alguma interface/operação provida por um componente seja opcional e o(s) componente(s) que a requer não seja(m) considerado(s) na referida iteração, para o uso das métricas DI e DO é como se todas as *features* opcionais da linha de produtos de microondas fossem escolhidas, e dessa forma, todas as interfaces e operações opcionais estivessem sendo usadas.

Tabela 9. Operações providas pelos componentes da LPS de microondas.

Interfaces	Operações	Componente que provê
IMWControl	sendControlRequest() alguns parâmetros - {features = TOD Clock, Minute Plus, Recipe e Power Level}	c₄
IHeatingElement	initialize()	c₅ e c₆
	startCooking(in level)	
	stopCooking()	
ILamp {feature = Light}	initialize()	c₇
	switchOn()	
	switchOff()	
ITurntable {feature = Turntable}	initialize()	c₈
	startTurning()	
	stopTurning()	
IBeeper	initialize()	c₉

{feature = Beeper}	beep()	
IDisplay	displayPrompt()	c_{10} e c_{11}
	displayTime()	
	clearScreen()	
	displayTOD() {feature = TOD Clock}	
IDisplayPrompt	initialize()	c_{12} , c_{13} , c_{14} , c_{15} e c_{16}
	read()	
	addPrompt()	
IOvenTimer	sendTimerRequest() alguns parâmetros - {features = Minute Plus e Recipe}	c_{17}
ICookingTimeData	clearCookingTime()	c_{18}
	updateCookingTime()	
	incrementCookingTime() {feature = Minute Plus}	
	decrementCookingTime()	
IRecipeData {feature = Recipe}	clearRecipe()	c_{18}
	selectRecipe()	
	nextStep()	
IWeightData {feature=Analog Weight}	clearWeight()	c_{18}
	setWeight()	
IPowerData {feature = Multi-Level Heating}	clearPower()	c_{18}
	setPower()	
	readPower ()	
ITODData {feature = TOD Clock}	clearTOD()	c_{18}
	updateTOD()	
	incrementTOD()	
ITODTimer {feature = TOD Clock}	startTOD()	c_{19}
	stopTOD()	
IRecipe	getRecipe()	c_{20}
	sendRecipe()	

A DSM da linha de produto construída com base nas métricas DO é descrita pela Figura 20, na qual os valores diferentes de zero foram sombreados para facilitar a visualização. A diferença mais significativa dessa DSM para a DSM construída com base na

métrica DI (Figura 13) diz respeito aos componentes *Display Interface – One Line* (c_{10}) e *Display Interface – Multi Line* (c_{11}), pois uma dentre as quatro operações definidas na interface provida por esses componentes (*IDisplay*) está habilitada somente para o componente c_{11} . Dessa forma, o cálculo da métrica DO é feito considerando que um componente que requer a interface *IDisplay* requer as quatro operações por ela definidas. O componente c_{10} provê apenas três operações dessa interface, enquanto que o componente c_{11} provê todas as quatro operações. Tomando como exemplo o componente *Microwave Oven Display* (c_4), que requer a interface *IDisplay* e o total de 33 operações, temos: $DO_{c_4, c_{10}} = 3/33 \cong 0,091$ e $DO_{c_4, c_{11}} = 4/33 \cong 0,121$.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
c0	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c1	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c2	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c3	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c4	1,000	1,000	1,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,083	0,000	0,000
c5	0,000	0,000	0,000	0,000	0,091	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c6	0,000	0,000	0,000	0,000	0,091	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c7	0,000	0,000	0,000	0,000	0,091	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c8	0,000	0,000	0,000	0,000	0,091	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c9	0,000	0,000	0,000	0,000	0,061	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c10	0,000	0,000	0,000	0,000	0,091	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,250	0,000	0,429	0,000
c11	0,000	0,000	0,000	0,000	0,121	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,333	0,000	0,571	0,000
c12	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c13	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c14	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c15	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c16	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c17	0,000	0,000	0,000	0,000	0,030	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c18	0,000	0,000	0,000	0,000	0,455	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,583	0,000	0,429	0,000
c19	0,000	0,000	0,000	0,000	0,061	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
c20	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,000	0,000

Figura 20. DSM da LPS de microondas segundo a métrica DO.

A DSM da linha de produto construída com base nas métricas DO e DF é descrita pela Figura 21, sendo utilizado $\alpha = 2/3$ na Equação (13). Nessa figura, as células tracejadas identificam as mudanças entre essa DSM e a DSM baseada nas métricas DI e DF descrita pela Figura 14. Conforme pode ser observado, as mudanças foram pequenas. Por exemplo, antes $DSM_{c_{18}, c_4} = [0,417 \times (2/3)] + [0,45 \times (1/3)] \cong 0,43$, e nessa nova DSM, $DSM_{c_{18}, c_4} = \left\{ \left(\frac{15}{33} \times \frac{2}{3} \right) + \left(0,45 \times \frac{1}{3} \right) \right\} \cong 0,45$, significando que de 33 operações requeridas pelo componente c_4 , 15 são providas pelo componente c_{18} . O valor do custo total de comunicação também sofreu uma pequena modificação, passando de 36,9 para 36,86.

A solução de agrupamento de componentes encontrada pelo algoritmo foi exatamente a mesma solução que foi encontrada a partir da DSM baseada nas métricas DI e DF (Figura

15), diferindo apenas quanto ao valor de CTC ($\cong 22,86$). Embora nesse projeto as soluções de agrupamento encontradas pelo uso das métricas DI e DO tenham sido equivalentes, nem sempre isso deve ocorrer, já que a métrica DO é mais precisa.

CTC $\approx 36,86$																					
	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
c0	0,00	0,17	0,17	0,07	0,04	0,17	0,17	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,05	0,00	0,00
c1	0,17	0,00	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,08	0,10	0,00	0,00
c2	0,17	0,11	0,00	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,08	0,04	0,00	0,00
c3	0,07	0,06	0,06	0,00	0,19	0,06	0,06	0,00	0,00	0,00	0,06	0,06	0,06	0,06	0,06	0,06	0,06	0,20	0,24	0,07	0,07
c4	0,70	0,70	0,70	0,85	0,00	0,03	0,07	0,04	0,04	0,04	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,17	0,15	0,04	0,04
c5	0,17	0,11	0,11	0,06	0,09	0,00	0,11	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,08	0,04	0,00	0,00
c6	0,17	0,11	0,11	0,06	0,13	0,11	0,00	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,08	0,10	0,00	0,00
c7	0,00	0,00	0,00	0,00	0,10	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c8	0,00	0,00	0,00	0,00	0,10	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c9	0,00	0,00	0,00	0,00	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c10	0,17	0,11	0,11	0,06	0,09	0,11	0,11	0,00	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,11	0,11	0,25	0,04	0,29	0,00
c11	0,17	0,11	0,11	0,06	0,11	0,11	0,11	0,00	0,00	0,00	0,11	0,00	0,11	0,11	0,11	0,11	0,11	0,31	0,04	0,38	0,00
c12	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,00	0,11	0,11	0,11	0,11	0,08	0,04	0,00	0,00
c13	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,11	0,00	0,11	0,11	0,11	0,08	0,04	0,00	0,00
c14	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,11	0,11	0,00	0,11	0,11	0,08	0,04	0,00	0,00
c15	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,11	0,11	0,11	0,00	0,11	0,08	0,04	0,00	0,00
c16	0,17	0,11	0,11	0,06	0,03	0,11	0,11	0,00	0,00	0,00	0,78	0,78	0,11	0,11	0,11	0,11	0,00	0,08	0,04	0,00	0,00
c17	0,11	0,08	0,08	0,20	0,13	0,08	0,08	0,00	0,00	0,00	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,00	0,14	0,00	0,00
c18	0,05	0,10	0,04	0,24	0,45	0,04	0,10	0,00	0,00	0,00	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,53	0,00	0,33	0,05
c19	0,00	0,00	0,00	0,07	0,08	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,05	0,00	0,00
c20	0,00	0,00	0,00	0,07	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,71	0,00	0,00

Figura 21. DSM da LPS de microondas segundo as métricas DO e DF.

A MBM associada à solução encontrada pelo algoritmo segundo a DSM baseada nas métricas DO e DF é descrita pela Figura 22. Novamente, como em todos os módulos há componentes relacionados ao componente *Microwave Oven Control* (c_4), pertencente ao módulo M_1 , todos os demais módulos se relacionam ao módulo M_1 , e com níveis de dependência inferiores às dependências internas aos módulos M_0 e M_1 . Apesar disso, o somatório das dependências internas aos módulos (1,43156) é superior ao somatório das dependências externas entre eles (0,66289), o que sinaliza que a solução é adequada.

	M_0	M_1	M_2	M_3	M_4
M_0	1,00000	0,34389	0,00000	0,00000	0,00000
M_1	0,29638	0,43156	0,00226	0,00226	0,00226
M_2	0,00000	0,00566	0,00000	0,00000	0,00000
M_3	0,00000	0,00452	0,00000	0,00000	0,00000
M_4	0,00000	0,00566	0,00000	0,00000	0,00000

Figura 22. MBM da LPS de microondas segundo as métricas DO e DF.

6.2. Linha de Produto de Comércio Eletrônico

A linha de produto de comércio eletrônico consiste em uma LPS altamente distribuída baseada na *World Wide Web* que controla sistemas B2B (*business-to-business*) e B2C

(*business-to-consumer*) [Gomaa 2004]. Trata-se de uma linha de produtos desenvolvida segundo um processo de engenharia reversa.

A fim de demonstrar o uso da presente abordagem para o desenvolvimento de um projeto em várias iterações, aqui é admitido que esta LPS será implementada em duas iterações: uma primeira iteração para implementar os componentes comuns e variantes, caracterizando a arquitetura de domínio, e uma segunda iteração para desenvolver as aplicações B2B e B2C, em conjunto. Para ambas as iterações, a DSM foi construída com base nas métricas DI e DF segundo a Equação (12), sendo $\alpha = 2/3$.

6.2.1. Primeira Iteração de Desenvolvimento

Os componentes da primeira iteração de desenvolvimento são identificados na arquitetura de domínio apresentada pela Figura 23, sendo os componentes variantes *Customer Interface* (c_0 e c_1), *Customer Agent* (c_2 e c_3), *Supplier Interface* (c_4 e c_5) e *Supplier Agent* (c_6 e c_7) detalhados na Figura 24. Na linha de produtos de E-Commerce, os componentes variantes são alternativos, o que significa que eles não podem coexistir em uma aplicação. Dessa forma, tais componentes não possuem dependência entre si por intermédio de interfaces/operações providas e requeridas. Apesar disso, como esses componentes estão relacionados ao mesmo ponto de variação, é esperado que possuam *features* em comum.

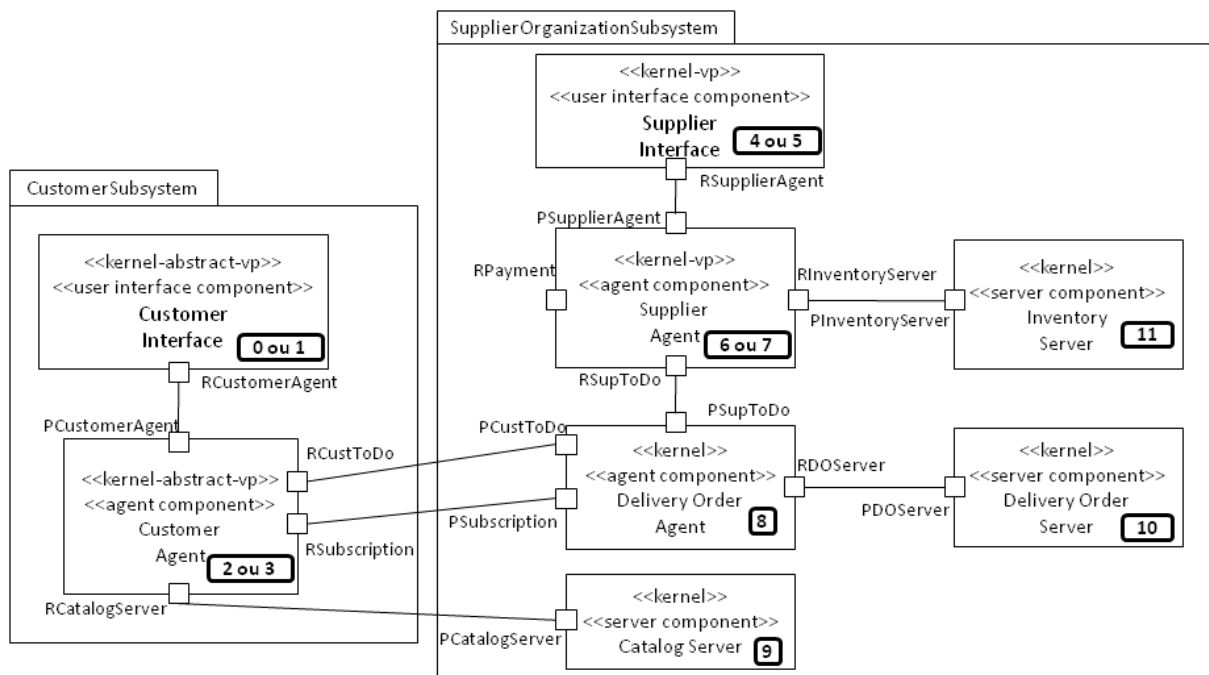


Figura 23. Arquitetura de domínio da LPS de E-Commerce [Gomaa 2004].

Inclusive, no caso particular do subsistema *Customer*, existe uma restrição de inclusão entre os componentes *B2B Customer Interface* (c_0) e *B2B Customer Agent* (c_2), bem como entre os componente *B2C Customer Interface* (c_1) e *B2C Customer Agent* (c_3). Restrição semelhante acontece no subsistema *Supplier Organization* entre os componentes *Supplier Interface* (c_4) e *Supplier Agent* (c_6), e os componentes *PO Supplier Interface* (c_5) e *PO Supplier Agent* (c_7). Restrições de inclusão entre componentes, por sua vez, são refletidas através de interfaces/operações providas e requeridas e *features*.

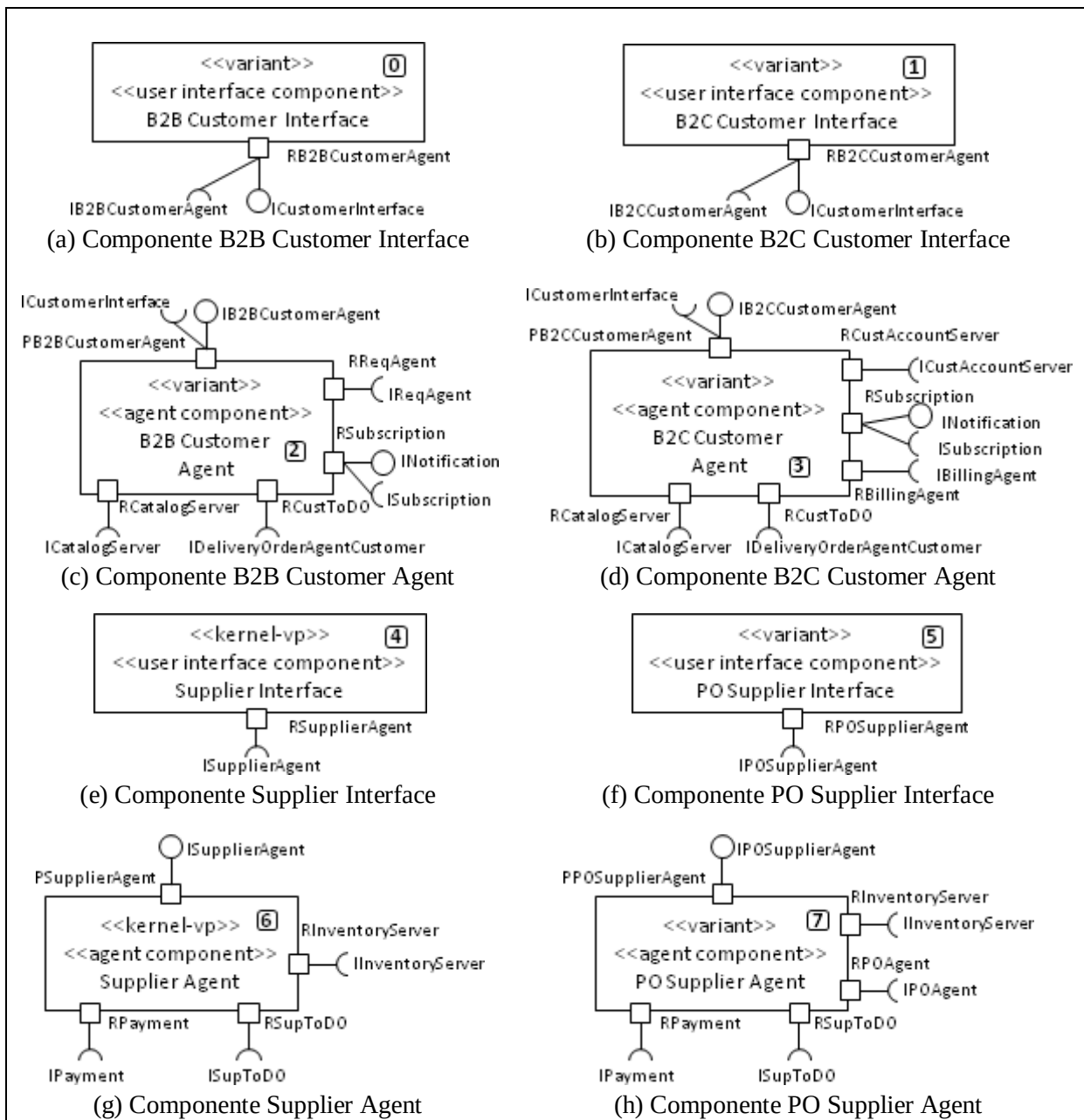


Figura 24. Visão detalhada dos componentes variantes [Gomaa 2004].

A informação sobre as *features* implementadas pelos componentes, requerida pela métrica DF, é apresentada na Tabela 10. Essa tabela contém todos os componentes da LPS de

E-Commerce, apesar dos componentes da segunda iteração de desenvolvimento serem detalhados mais adiante. De acordo com essa tabela, os componentes variantes da Figura 24 possuem os seguintes valores de DF em cada ponto de variação: $DO_{c_0, c_1} = 1/3$, $DO_{c_2, c_3} = 1/3$, $DO_{c_4, c_5} = 1/2$ e $DO_{c_6, c_7} = 1/2$.

Tabela 10. Relação de *features* por componente na LPS de E-Commerce.

Índice	Componente	Feature	Quantidade
c_0	B2BCustomerInterface	E-commerce kernel	2
		Business Customer	
c_1	B2CCustomerInterface	E-commerce kernel	2
		Home Customer	
c_2	B2BCustomerAgent	E-commerce kernel	2
		Business Customer	
c_3	B2CCustomerAgent	E-commerce kernel	2
		Home Customer	
c_4	SupplierInterface	E-commerce kernel	1
c_5	POSupplierInterface	E-commerce kernel	2
		Purchase Order	
c_6	SupplierAgent	E-commerce kernel	1
c_7	POSupplierAgent	E-commerce kernel	2
		Purchase Order	
c_8	DeliveryOrderAgent	E-commerce kernel	1
c_9	CatalogServer	E-commerce kernel	1
c_{10}	DeliveryOrderServer	E-commerce kernel	1
c_{11}	InventoryServer	E-commerce kernel	1
c_{12}	PurchaseOrderAgent	Purchase Order	1
c_{13}	WholesalerInterface	Purchase Order	1
c_{14}	PurchaseOrderServer	Purchase Order	1
c_{15}	BankInterface	Bank	1
c_{16}	BillingAgent	Home Customer	1
c_{17}	CustomerAccountServer	Home Customer	1
c_{18}	RequisitionAgent	Business Customer	1
c_{19}	InvoiceAgent	Business Customer	1
c_{20}	InvoiceServer	Business Customer	1
c_{21}	RequisitionServer	Business Customer	1

c_{22}	OperationFoundsServer	Business Customer	1
c_{23}	ContractsServer	Business Customer	1
c_{24}	AccountsPayableServer	Business Customer	1
c_{25}	AuthorizationCenterInterface	Home Customer	1

A Figura 25 ilustra a DSM da primeira iteração de desenvolvimento. A fim de demonstrar como os valores foram definidos na matriz, considere os componentes *B2C Customer Interface* (c_1) e *B2C Customer Agent* (c_3). Conforme pode ser visto pelas Figura 24d e 22b respectivamente, c_3 requer 6 interfaces e c_1 provê 1 delas. Então, $DIC_{c_3, c_1} = 1/6 \cong 0,17$. Além disso, c_1 e c_3 implementam as mesmas *features*, o que significa que $FDC_{c_3, c_1} = \frac{2}{2} = 1$. Nesse sentido, a dependência do componente c_3 para o componente c_1 é $DSMc_{c_1, c_3} = [(0,17 \times 2/3) + (1 \times 1/3)] \cong 0,444$.

$CTC \cong 31,86$												
	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
c_0	0,00	0,11	0,47	0,11	0,17	0,11	0,17	0,11	0,17	0,17	0,17	0,17
c_1	0,11	0,00	0,11	0,44	0,17	0,11	0,17	0,11	0,17	0,17	0,17	0,17
c_2	1,00	0,11	0,00	0,11	0,17	0,11	0,17	0,11	0,39	0,17	0,17	0,17
c_3	0,11	1,00	0,11	0,00	0,17	0,11	0,17	0,11	0,39	0,17	0,17	0,17
c_4	0,17	0,17	0,17	0,17	0,00	0,17	0,33	0,17	0,33	0,33	0,33	0,33
c_5	0,11	0,11	0,11	0,11	0,17	0,00	0,17	0,33	0,17	0,17	0,17	0,17
c_6	0,17	0,17	0,17	0,17	1,00	0,17	0,00	0,17	0,33	0,33	0,33	0,33
c_7	0,11	0,11	0,11	0,11	0,17	1,00	0,17	0,00	0,17	0,17	0,17	0,17
c_8	0,17	0,17	0,43	0,39	0,33	0,17	0,56	0,33	0,00	0,33	0,33	0,33
c_9	0,17	0,17	0,30	0,28	0,33	0,17	0,33	0,17	0,33	0,00	0,33	0,33
c_{10}	0,17	0,17	0,17	0,17	0,33	0,17	0,33	0,17	0,56	0,33	0,00	0,33
c_{11}	0,17	0,17	0,17	0,17	0,33	0,17	0,56	0,33	0,33	0,33	0,33	0,00

Figura 25. DSM da primeira iteração da LPS de E-Commerce.

A Figura 26 mostra a CMM da solução encontrada pelo algoritmo. Com base nessa CMM, a Figura 27 apresenta uma versão reordenada da DSM original, que melhor identifica os agrupamentos (células sombreadas). A solução de agrupamento encontrada corresponde à solução ótima, o que mostra a eficiência do algoritmo desenvolvido.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
M_0	0	0	0	0	1	0	1	0	1	1	1	1
M_1	0	1	0	1	0	1	0	1	0	0	0	0
M_2	1	0	1	0	0	0	0	0	0	0	0	0

Figura 26. CMM da primeira iteração da LPS de E-Commerce.

De acordo com as células destacadas em negrito na Figura 27, é possível observar que os componentes *Supplier Interface* (c_4) e *Supplier Agent* (c_6) são bastante dependentes entre si, assim como os componentes *B2C Customer Interface* (c_1) e *B2C Customer Agent* (c_3), os componentes *PO Supplier Interface* (c_5) e *PO Supplier Agent* (c_7), e os componentes *B2B Customer Interface* (c_0) e *B2B Customer Agent* (c_2), o que está coerente com o projeto arquitetural.

$CTC \cong 22,57$													
		c_4	c_6	c_8	c_9	c_{10}	c_{11}	c_1	c_3	c_5	c_7	c_0	c_2
M_0	c_4	0,0	0,33	0,33	0,33	0,33	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_6	1,0	0,0	0,33	0,33	0,33	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_8	0,33	0,56	0,0	0,33	0,33	0,33	0,17	0,39	0,17	0,33	0,17	0,43
	c_9	0,33	0,33	0,33	0,0	0,33	0,33	0,17	0,28	0,17	0,17	0,17	0,3
	c_{10}	0,33	0,33	0,56	0,33	0,0	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_{11}	0,33	0,56	0,33	0,33	0,33	0,0	0,17	0,17	0,17	0,33	0,17	0,17
M_1	c_1	0,17	0,17	0,17	0,17	0,17	0,17	0,0	0,44	0,11	0,11	0,11	0,11
	c_3	0,17	0,17	0,39	0,17	0,17	0,17	1,0	0,0	0,11	0,11	0,11	0,11
	c_5	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,0	0,33	0,11	0,11
	c_7	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	1,0	0,0	0,11	0,11
M_2	c_0	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,11	0,11	0,0	0,47
	c_2	0,17	0,17	0,39	0,17	0,17	0,17	0,11	0,11	0,11	0,11	1,0	0,0

Figura 27. DSM da primeira iteração da LPS de E-Commerce reordenada.

Partindo disso, de acordo com a Figura 27 também é possível observar que os componentes *Supplier Interface* (c_4), *Supplier Agent* (c_6), *Delivery Order Agent* (c_8), *Catalog Server* (c_9), *Delivery Order Server* (c_{10}) e *Inventory Server* (c_{11}) foram agrupados juntos, em concordância com a definição do subsistema *Supplier Organization*, mas que os componentes *PO Supplier Interface* (c_5) e *PO Supplier Agent* (c_7) não foram incluídos. Como agrupar todos esses componentes resultaria em um módulo grande, eles foram separados, sendo preservados os relacionamentos de interdependência, que são os mais expressivos. Dessa forma, dado que os componentes c_4 e c_6 devem ficar juntos, assim como os componentes c_5 e c_7 , e que $DSMc_{11,c_6} = 0,56$ é mais expressivo do que $DSMc_{11,c_7} = 0,33$, c_{11} é agrupado com os componentes c_4 e c_6 . Daí, como $DSMc_{8,c_6} = 0,56$ é mais expressivo do que $DSMc_{8,c_7} = 0,33$, c_8 é agrupado com c_4 , c_6 e c_{11} . O componente c_9 , apesar de estar relacionado aos componentes *B2B Customer Agent* (c_2) e *B2C Customer Agent* (c_3) através de interfaces providas e requeridas, também é adicionado ao módulo M_0 , porque de acordo com a métrica FD, c_9 está mais relacionado com os componentes em M_0 do que com c_2 e c_3 , o que não é visualmente perceptível na Figura 23.

Dada a forte dependência entre os componentes *B2B Customer Interface* (c_0) e *B2B Customer Agent* (c_2), em conformidade à restrição de inclusão existente entre eles, é compreensível a definição do módulo M_2 , o que também está coerente com a definição do subsistema *Customer*.

Já o entendimento sobre a razão dos componentes *B2C Customer Interface* (c_1), *B2C Customer Agent* (c_3), *PO Supplier Interface* (c_5) e *PO Supplier Agent* (c_7) terem sido agrupados juntos requer a análise de resultados alternativos. Conforme anteriormente analisado, os componentes c_1 e c_3 são mais dependentes entre si, assim como os componentes c_5 e c_7 . Dessa forma, um resultado alternativo poderia ser organizar esses componentes em dois módulos distintos, resultando na DSM reordenada descrita pela Figura 28.

$CTC \cong 22,7$													
		c_4	c_6	c_8	c_9	c_{10}	c_{11}	c_1	c_3	c_5	c_7	c_0	c_2
M_0	c_4	0,00	0,33	0,33	0,33	0,33	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_6	1,00	0,00	0,33	0,33	0,33	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_8	0,33	0,56	0,00	0,33	0,33	0,33	0,17	0,39	0,17	0,33	0,17	0,43
	c_9	0,33	0,33	0,33	0,00	0,33	0,33	0,17	0,28	0,17	0,17	0,17	0,30
	c_{10}	0,33	0,33	0,56	0,33	0,00	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_{11}	0,33	0,56	0,33	0,33	0,33	0,00	0,17	0,17	0,17	0,33	0,17	0,17
M_1	c_1	0,17	0,17	0,17	0,17	0,17	0,17	0,00	0,44	0,11	0,11	0,11	0,11
	c_3	0,17	0,17	0,39	0,17	0,17	0,17	1,00	0,00	0,11	0,11	0,11	0,11
M_2	c_5	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,00	0,33	0,11	0,11
	c_7	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	1,00	0,00	0,11	0,11
M_3	c_0	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,11	0,11	0,00	0,47
	c_2	0,17	0,17	0,39	0,17	0,17	0,17	0,11	0,11	0,11	0,11	1,00	0,00

Figura 28. DSM da primeira solução alternativa para a LPS de E-Commerce reordenada.

A comparação entre o custo da solução da Figura 28 e a solução reportada na Figura 27 mostra que a primeira aumenta o valor de CCc_3 de 1,79 para 1,86 e de CCc_7 de 1,74 para 1,84, ao mesmo tempo que reduz o valor de CCc_5 de 1,65 para 1,63, implicando em um valor superior de CTC (de 22,57 para 22,7).

Como os componentes c_0 e c_2 são alternativos aos componentes c_1 e c_3 , dado que c_0 requer c_2 e c_1 requer c_3 , outro resultado possível seria substituir os componentes c_1 e c_3 no módulo M_1 na Figura 27 pelos componentes c_0 e c_2 , deixando c_1 e c_3 em M_2 , resultando na DSM reordenada da Figura 29.

$CTC \cong 22,576$													
		c_4	c_6	c_8	c_9	c_{10}	c_{11}	c_1	c_3	c_0	c_2	c_5	c_7
M_0	c_4	0,00	0,33	0,33	0,33	0,33	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_6	1,00	0,00	0,33	0,33	0,33	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_8	0,33	0,56	0,00	0,33	0,33	0,33	0,17	0,39	0,17	0,43	0,17	0,33
	c_9	0,33	0,33	0,33	0,00	0,33	0,33	0,17	0,28	0,17	0,30	0,17	0,17
	c_{10}	0,33	0,33	0,56	0,33	0,00	0,33	0,17	0,17	0,17	0,17	0,17	0,17
	c_{11}	0,33	0,56	0,33	0,33	0,33	0,00	0,17	0,17	0,17	0,17	0,17	0,33
M_1	c_1	0,17	0,17	0,17	0,17	0,17	0,17	0,00	0,44	0,11	0,11	0,11	0,11
	c_3	0,17	0,17	0,39	0,17	0,17	0,17	1,00	0,00	0,11	0,11	0,11	0,11
M_2	c_0	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,00	0,47	0,11	0,11
	c_2	0,17	0,17	0,39	0,17	0,17	0,17	0,11	0,11	1,00	0,00	0,11	0,11
	c_5	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,11	0,11	0,00	0,33
	c_7	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,11	0,11	1,00	0,00

Figura 29. DSM da segunda solução alternativa para a LPS de E-Commerce reodenada.

Novamente, a comparação entre o custo da solução da Figura 29 e a solução da Figura 27 mostra que na primeira o valores de CCc_0 e CCc_3 são maiores, apesar dos valores de CCc_1 e CCc_2 serem menores, resultando num valor final de CTC ligeiramente superior (de 22,571 para 22,576). Dado que a diferença dos valores de CTC é muito sutil, alguém ainda pode questionar o fato de ser afirmado que a solução da Figura 27 é melhor do que a solução da Figura 29. Além disso, na perspectiva do significado da arquitetura, alguém pode considerar mais adequada a solução representada pela Figura 28, dado que os componentes em M_1 possuem restrição de inclusão entre si, assim como os componentes em M_2 e em M_3 , além dos componentes em cada um desses módulos pertencerem ao mesmo subsistema. A fim de esclarecer qual é realmente a melhor solução dentre as três soluções discutidas é necessário comparar a MBM associada a cada uma delas.

A MBM relacionada à solução encontrada pelo algoritmo (Figura 27) é apresentada na Figura 30. Nessa matriz, a dependência interna do módulo M_0 consiste na soma de todas as células na primeira área sombreada da Figura 27: $MBM'_{M_0, M_0} = [(0,33 \times 26) + (0,56 \times 3) + 1] = 11,26$. Uma vez que todas as células são normalizadas e $MBM_{\max} = 11,26$, o valor final de MBM_{M_0, M_0} é 1. Igualmente, a dependência externa do módulo M_0 para o módulo M_1 é o somatório de todas as células abaixo da primeira área sombreada e à esquerda da segunda área sombreada na Figura 27. Daí, temos: $MBM'_{M_0, M_1} = [(0,17 \times 23) + 0,39] \cong 4,3$ e $MBM_{M_0, M_1} = \frac{4,3}{11,26} \cong 0,382$.

	M_0	M_1	M_2
M_0	1,00000	0,42007	0,21581
M_1	0,38188	0,32416	0,07815
M_2	0,20071	0,07815	0,13055

Figura 30. MBM da primeira iteração da LPS de E-Commerce.

Na Figura 30, a primeira célula sombreada representa a dependência interna ao módulo M_0 e a linha e a coluna da matriz correspondentes à M_0 expressam as dependências externas a esse módulo. Observe que esses valores são inferiores ao valor da dependência interna de M_0 , o máximo valor nessa MBM. Igualmente, a segunda célula sombreada identifica a dependência interna ao módulo M_1 , e a linha e a coluna da matriz correspondentes à M_1 identificam as dependências externas a esse módulo.

Dado que os módulos M_0 e M_1 são interdependentes, os valores de dependência entre esses componentes são inferiores à dependência interna de M_0 , mas ligeiramente superiores à dependência interna de M_1 . Situação semelhante ocorre em relação ao módulo M_2 , cujo valor de dependência interna é identificado pela terceira célula sombreada na Figura 30. As dependências entre os módulos M_2 e M_1 são inferiores à dependência interna de M_2 , mas as dependências entre M_2 e M_0 são ligeiramente superiores à dependência interna de M_2 . Apesar disso, o somatório das dependências internas aos módulos (1,45471) é superior ao somatório das dependências externas entre eles (1,37477), o que expressa que a solução é adequada.

Comparando a MBM da Figura 30 com a MBM da Figura 31, associada à solução alternativa da Figura 28, vê-se que a MBM da Figura 30 é melhor na perspectiva da maximização das dependências internas aos módulos e minimização das dependências externas, pois na MBM da **Error! Reference source not found.** o somatório das dependências internas aos módulos é 1,37656 e o somatório das dependências externas entre eles é 1,45295, significando que o total de dependências internas é inferior ao total de dependências externas, o que não é desejável para os propósitos da presente abordagem. Isso evidencia que a solução encontrada pelo algoritmo (Figura 27) é mais adequada do que a solução alternativa da Figura 28.

	M_0	M_1	M_2	M_3
M_0	1,00000	0,20959	0,21581	0,21048
M_1	0,18117	0,11812	0,03908	0,03908
M_2	0,20071	0,03908	0,13055	0,03908
M_3	0,20071	0,03908	0,03908	0,12789

Figura 31. MBM da primeira solução alternativa para a LPS de E-Commerce.

Em se tratando da MBM associada à solução alternativa da Figura 29, descrita pela Figura 32, o somatório das dependências internas aos módulos e o somatório das dependências externas entre módulos resultam exatamente nos mesmos valores calculados para a MBM da Figura 30 associada à solução da Figura 27. Isso indica que tais soluções são equivalentes na perspectiva de maximização das dependências internas aos módulos e minimização das dependências entre módulos, embora se diferenciem pelos valores de dependências internas e externas aos módulos M_1 e M_2 . Nessa situação, o uso da abordagem possibilita que o gerente de projeto realize uma avaliação comparativa entre soluções de agrupamento, estando coerente com o propósito de prover recomendações.

	M_0	M_1	M_2
M_0	1,00000	0,42540	0,21048
M_1	0,38188	0,32682	0,07815
M_2	0,20071	0,07815	0,12789

Figura 32. MBM da segunda solução alternativa para a LPS de E-Commerce.

6.2.2. Segunda Iteração de Desenvolvimento

Na segunda iteração de desenvolvimento, 14 componentes foram adicionados, os quais em sua maioria são identificados na arquitetura das aplicações B2B e B2C apresentadas pela Figura 33 e pela Figura 34, respectivamente. Em ambas as aplicações foram utilizados os componentes *Supplier Interface* (c_4) e *Supplier Agent* (c_6). Dessa forma, os componentes alternativos a estes *PO Supplier Interface* (c_5) e *PO Supplier Agent* (c_7), e os demais componentes a eles associados, *Purchase Order Agent* (c_{12}), *Wholesaler Interface* (c_{13}) e *Purchase Order Server* (c_{14}), são apresentados na Figura 35.

Na arquitetura da aplicação B2B, é utilizado o subsistema *Business Customer Organization*, que é bem mais complexo do que o subsistema *Home Customer*. Esse subsistema, além de conter os componentes *B2B Customer Interface* (c_0) e *B2B Customer Agent* (c_2), contém os componentes opcionais *Requisition Agent* (c_{18}), *Invoice Agent* (c_{19}), *Invoice Server* (c_{20}), *Requisition Server* (c_{21}), *Operation Funds Server* (c_{22}), *Contracts Server* (c_{23}), e *Accounts Payable Server* (c_{24}).

Já na arquitetura da aplicação B2C, o subsistema *Customer* é constituído pelos componentes *B2C Customer Interface* (c_1) e *B2C Customer Agent* (c_3). O componente c_3 requer os componentes opcionais *Billing Agent* (c_{16}) e *Customer Account Server* (c_{17}) que fazem parte do subsistema *Supplier Organization*. Por sua vez, o componente c_{16} requer o componente *Authorization Center Interface* (c_{25}).

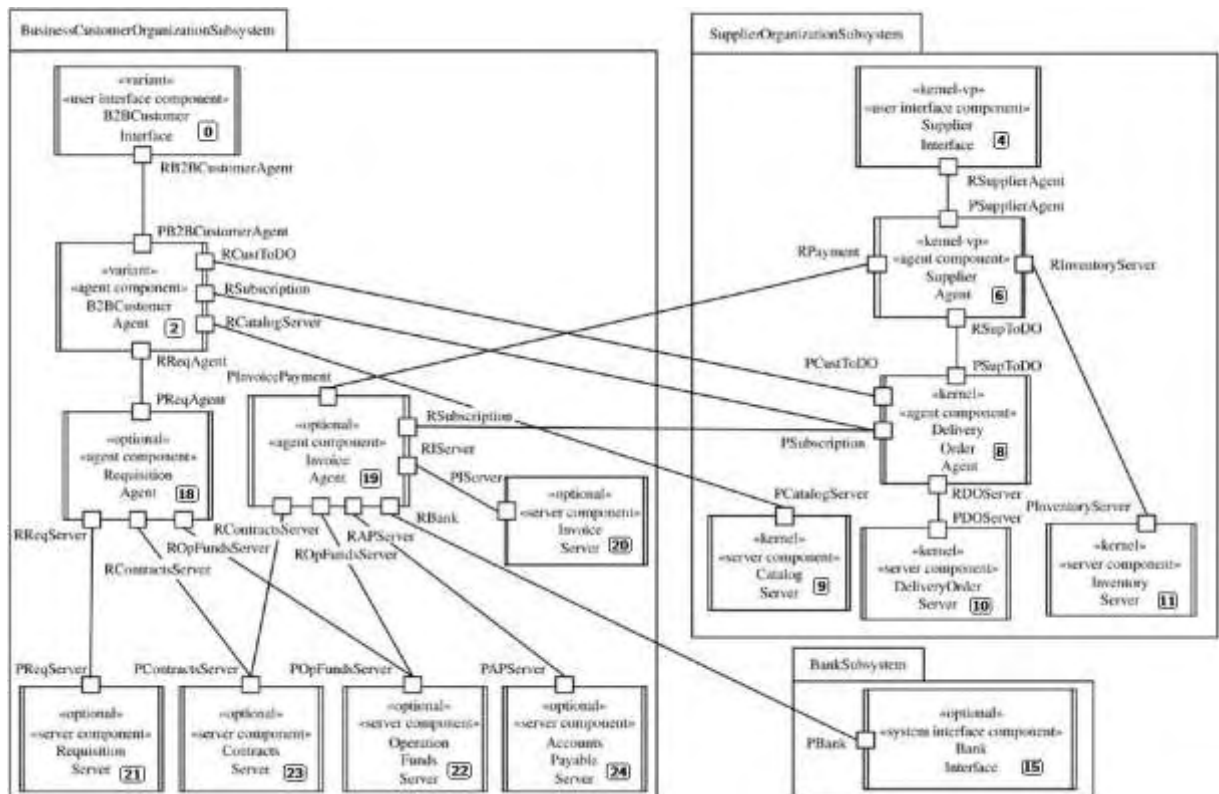


Figura 33. Arquitetura da aplicação B2B [Gomaa 2004].

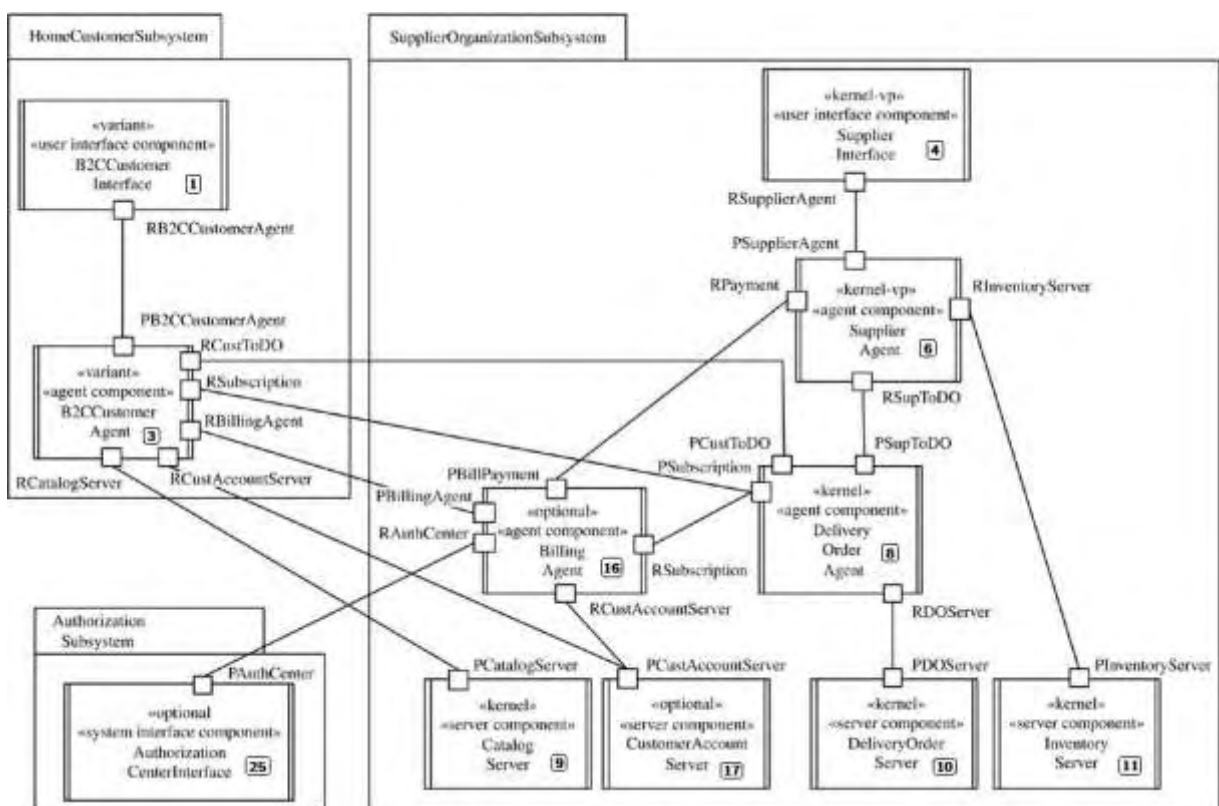


Figura 34. Arquitetura da aplicação B2C [Gomaa 2004].

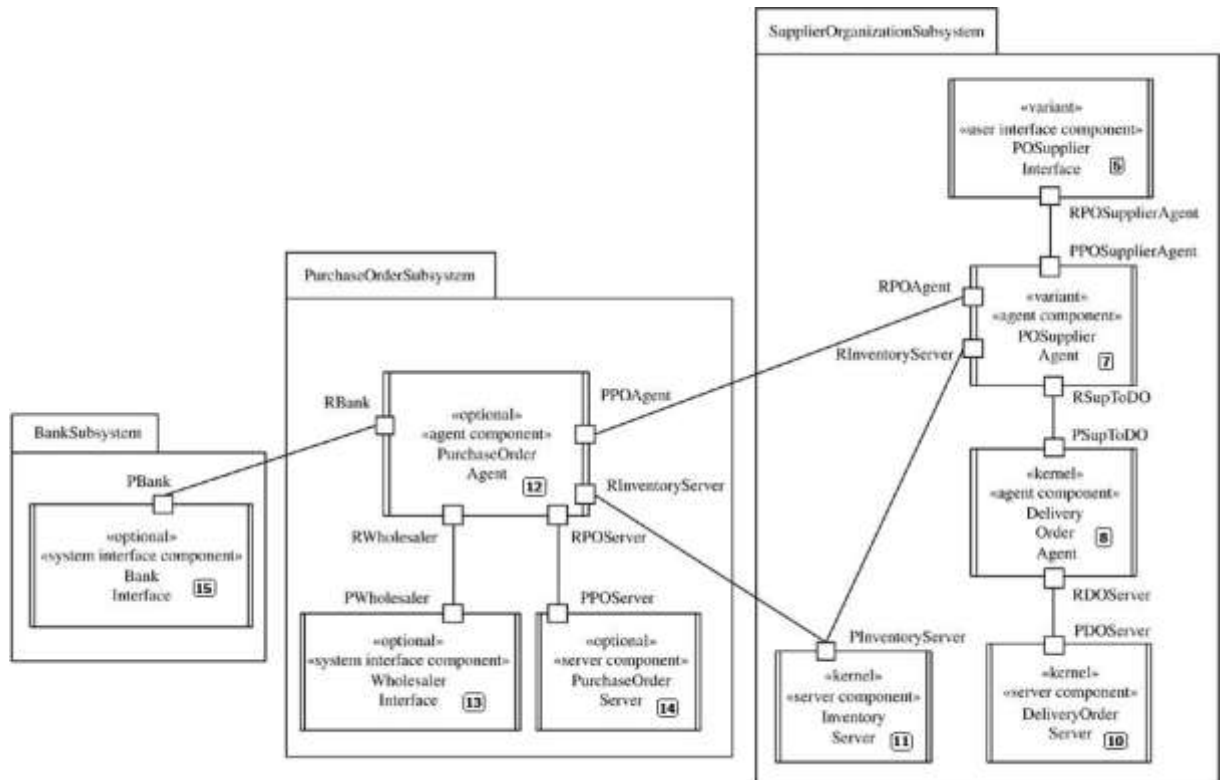


Figura 35. Componentes opcionais da LPS de E-Commerce [Gomaa 2004].

A DSM da segunda iteração é definida pela Figura 36, na qual as dependências externas aos módulos e diferentes de zero foram destacadas em negrito para facilitar a visualização.

		CTC ≈ 49,14																									
		c4	c6	c8	c9	c10	c11	c1	c3	c5	c7	c0	c2	c12	c13	c14	c15	c16	c17	c18	c19	c20	c21	c22	c23	c24	c25
M0	c4	0,00	0,33	0,33	0,33	0,33	0,33	0,17	0,17	0,17	0,17	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	c6	1,00	0,00	0,33	0,33	0,33	0,33	0,17	0,17	0,17	0,17	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	c8	0,33	0,56	0,00	0,33	0,33	0,33	0,17	0,39	0,17	0,33	0,17	0,43	0,00	0,00	0,00	0,00	0,22	0,00	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c9	0,33	0,33	0,33	0,00	0,33	0,33	0,17	0,28	0,17	0,17	0,17	0,30	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	c10	0,33	0,33	0,56	0,33	0,00	0,33	0,17	0,17	0,17	0,17	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M1	c11	0,33	0,56	0,33	0,33	0,33	0,00	0,17	0,17	0,17	0,33	0,17	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	c1	0,17	0,17	0,17	0,17	0,17	0,17	0,00	0,44	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,17	
	c3	0,17	0,17	0,39	0,17	0,17	0,17	1,00	0,00	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,17	
	c5	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,00	0,33	0,11	0,11	0,17	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	
M2	c7	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	1,00	0,00	0,11	0,11	0,17	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	c0	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,11	0,11	0,00	0,47	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,00	
M3-M16	c2	0,17	0,17	0,39	0,17	0,17	0,17	0,11	0,11	0,11	0,11	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,00
	M3	c12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,33	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	M4	c13	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,50	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	M5	c14	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,50	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	M6	c15	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	M7	c16	0,00	0,22	0,22	0,00	0,00	0,00	0,17	0,28	0,00	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,33
	M8	c17	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,28	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,56	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33
	M9	c18	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,30	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,00
	M10	c19	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,33	0,33	0,33	0,33	0,00
	M11	c20	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,44	0,00	0,33	0,33	0,33	0,00
	M12	c21	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,56	0,33	0,33	0,00	0,33	0,33	0,00
	M13	c22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,56	0,44	0,33	0,33	0,00	0,33	0,00
	M14	c23	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,56	0,44	0,33	0,33	0,33	0,00	0,33
	M15	c24	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,44	0,33	0,33	0,33	0,33	0,00
	M16	c25	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,56	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Figura 36. DSM da segunda iteração da LPS de E-Commerce.

A DSM da Figura 36 contém os componentes da primeira iteração de desenvolvimento ($c_0 - c_{11}$), pertencentes aos módulos M_0 , M_1 e M_2 , e os novos componentes ($c_{12} - c_{25}$). As células nessa DSM foram reordenadas para refletir tais módulos previamente definidos. Dado que os componentes da primeira iteração já foram agrupados, na segunda iteração eles não podem ser deslocados para outros módulos, mas novos componentes podem ser adicionados aos seus módulos. Já os novos componentes estão em módulos isolados, a priori.

A solução de agrupamento encontrada pelo algoritmo para a segunda iteração de desenvolvimento é descrita pela DSM reordenada da Figura 37, na qual os módulos previamente definidos foram tracejados. Conforme pode ser observado nessa figura, os componentes *B2B Customer Interface* (c_0), *B2B Customer Agent* (c_2), *Requisition Agent* (c_{18}), *Invoice Agent* (c_{19}), *Invoice Server* (c_{20}), *Requisition Server* (c_{21}), *Operation Funds Server* (c_{22}), *Contracts Server* (c_{23}), e *Accounts Payable Server* (c_{24}) foram agrupados juntos (módulo M_0), o que está coerente com o subsistema *Business Customer Organization* descrito pela Figura 33.

		CTC = 31,69																											
		c0	c2	c18	c19	c20	c21	c22	c23	c24	c1	c3	c5	c7	c16	c17	c25	c4	c6	c8	c9	c10	c11	c12	c13	c14	c15		
M0	c0	0,00	0,47	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,00	0,00	0,00	0,00		
	c2	1,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,17	0,17	0,39	0,17	0,17	0,17	0,00	0,00	0,00	0,00		
	c18	0,17	0,30	0,00	0,33	0,33	0,33	0,33	0,33	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
	c19	0,17	0,17	0,33	0,00	0,33	0,33	0,33	0,33	0,33	0,00	0,00	0,00	0,17	0,00	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
	c20	0,17	0,17	0,33	0,44	0,00	0,33	0,33	0,33	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
	c21	0,17	0,17	0,56	0,33	0,33	0,00	0,33	0,33	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
	c22	0,17	0,17	0,56	0,44	0,33	0,33	0,00	0,33	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
	c23	0,17	0,17	0,56	0,44	0,33	0,33	0,33	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
M1	c24	0,17	0,17	0,33	0,44	0,33	0,33	0,33	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
	c1	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,44	0,11	0,11	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,00	0,00	0,00			
	c3	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00	0,00	0,11	0,11	0,17	0,17	0,17	0,17	0,17	0,39	0,17	0,17	0,17	0,00	0,00	0,00			
	c5	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,11	0,11	0,00	0,33	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,00			
	c7	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,11	0,11	1,00	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,17	0,00			
	c16	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,28	0,00	0,17	0,00	0,33	0,33	0,00	0,22	0,22	0,00	0,00	0,00	0,00	0,00	0,00			
	c17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,28	0,00	0,00	0,56	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
	c25	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,00	0,00	0,00	0,56	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00			
M2	c4	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,00	0,00	0,00	0,00	0,33	0,33	0,33	0,33	0,33	0,00	0,00	0,00			
	c6	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,00	0,00	0,00	1,00	0,00	0,33	0,33	0,33	0,33	0,00	0,00	0,00			
	c8	0,17	0,43	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,17	0,39	0,17	0,33	0,22	0,00	0,00	0,33	0,56	0,00	0,33	0,33	0,33	0,00	0,00	0,00			
	c9	0,17	0,30	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,28	0,17	0,17	0,00	0,00	0,00	0,33	0,33	0,33	0,00	0,33	0,33	0,00	0,00	0,00			
	c10	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,17	0,17	0,00	0,00	0,00	0,33	0,33	0,56	0,33	0,00	0,33	0,00	0,00	0,00			
	c11	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,17	0,33	0,00	0,00	0,00	0,33	0,56	0,33	0,33	0,33	0,00	0,17	0,00	0,00			
M3	c12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17			
	c13	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,50	0,00	0,33			
	c14	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,50	0,33	0,00			
	c15	0,00	0,00	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,00	0,00			

Figura 37. DSM que descreve a solução da segunda iteração da LPS de E-Commerce.

O módulo M_1 contém os componentes previamente agrupados *B2C Customer Interface* (c_1), *B2C Customer Agent* (c_3), *PO Supplier Interface* (c_5) e *PO Supplier Agent* (c_7), além dos componentes *Billing Agent* (c_{16}), *Customer Account Server* (c_{17}) e *Authorization Center Interface* (c_{25}). Os componentes c_5 , c_7 , c_{16} e c_{17} pertencem ao subsistema *Supplier*

Organization, enquanto que os componentes c_1 e c_3 pertencem ao subsistema *Home Customer*, e o componente c_{25} pertence ao subsistema *Authorization*. Apesar disso, tais componentes são dependentes entre si, sendo mais expressivas as dependências entre os componentes c_{16} , c_{17} e c_{25} . Isso pode ser verificado tanto nos valores da DSM, quanto no projeto arquitetural segundo a Figura 34.

Ainda na Figura 37, nota-se que o módulo M_2 não sofreu alteração alguma, correspondendo ao módulo M_0 da DSM da Figura 36, enquanto que o módulo M_3 , que contém os componentes *Purchase Order Agent* (c_{12}), *Wholesaler Interface* (c_{13}) e *Purchase Order Server* (c_{14}), está coerente com o subsistema *Purchase Order* descrito na Figura 35. O componente *Bank Interface* (c_{15}), apesar de não fazer parte do subsistema *Purchase Order*, também foi incluído no módulo M_3 , uma vez que só mantém relacionamento de dependência com os componentes c_{12} (0,17) e *Invoice Agent* (c_{19}) (0,12), prevalecendo o relacionamento de maior nível de dependência.

A MBM relacionada à solução encontrada pelo algoritmo é apresentada na Figura 38. Conforme pode ser verificado, as dependências externas aos módulos são predominantemente inferiores às dependências internas aos módulos, característica desejável nas soluções de agrupamento. A exceção são os valores destacados em negrito, que caracterizam dependências externas que são superiores somente à dependência interna do módulo M_3 . Entretanto, dado que esses valores configuram dependências externas ao módulo M_2 e não ao módulo M_3 , e que o somatório das dependências externas entre os módulos (0,89555) é inferior ao somatório das dependências internas (2,02014), a solução ainda pode ser considerada uma boa solução de agrupamento.

	M_0	M_1	M_2	M_3
M_0	1,00000	0,04918	0,11616	0,00000
M_1	0,04122	0,39110	0,22201	0,04778
M_2	0,11897	0,23185	0,52740	0,00796
M_3	0,00515	0,05527	0,00000	0,10164

Figura 38. MBM da segunda iteração de desenvolvimento.

6.3. Multi-MOM

O Multi-MOM consiste numa linha de produtos de *middleware* para computação móvel que está essencialmente focada na funcionalidade de comunicação. A Figura 39 descreve a arquitetura de domínio desta LPS, segundo a qual há um único ponto de variação,

representado pelo componente genérico *Communication Paradigm*, havendo cinco componentes variantes que não são excludentes entre si ($c_7 - c_{11}$).

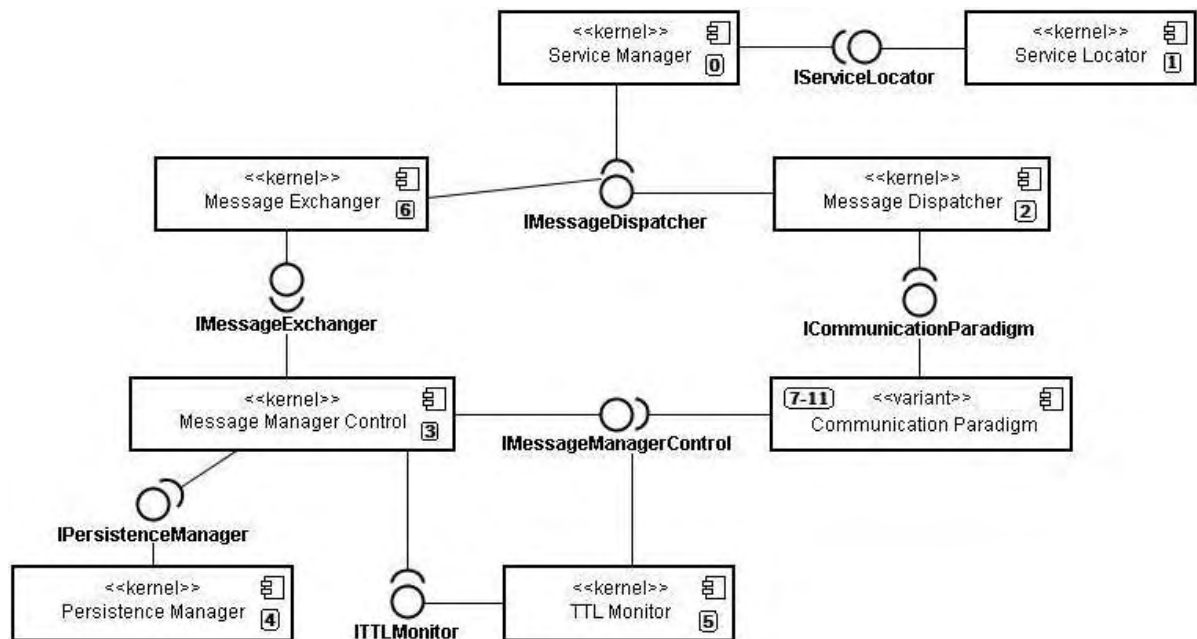


Figura 39. Arquitetura de domínio do Multi-MOM.

De acordo com a Figura 39 e com a Tabela 11, que apresenta as *features* contempladas por cada componente conforme definido pelo arquiteto do projeto, foram calculadas as métricas DI e DF para essa LPS. Dessa forma, foi usada a Equação (12), sendo $\alpha = 2/3$, tal como nos estudos de caso anteriormente apresentados.

Tabela 11. Relação de *features* por componente definida pelo arquiteto do projeto.

Índice	Componente	Feature	Quantidade
c_0	Service Manager	Localização de serviços	1
c_1	Service Locator	Localização de serviços	1
c_2	Message Dispatcher	Reflexão	1
c_3	Message Manager Control	Buffer de envio	1
c_4	Persistence Manager	Persistência de mensagens	1
c_5	TTL Monitor	Monitor de TTL	1
c_6	Message Exchanger	Troca de mensagens pela rede	1
c_7	Communication Paradigm-Publish/Subscribe	Publish/Subscribe	2
		Buffer de envio	
c_8	Communication Paradigm-Message Queue	Message Queue	2
		Buffer de envio	
c_9	Communication Paradigm-Tuple Space	Tuple Space	2
		Buffer de envio	
c_{10}	Communication Paradigm-PTP	PTP Notification	2

	Notification	Buffer de envio	
c_{11}	Communication Paradigm-Synchronous	Synchronous	2
		Buffer de envio	

Com base no cálculo das métricas DI e DF, foi construída a DSM descrita pela Figura 40, na qual os valores diferentes de zero foram destacados em negrito para facilitar a visualização.

$CTC \cong 13,88$												
	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
c_0	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_1	0,67	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_2	0,33	0,00	0,00	0,00	0,00	0,00	0,67	0,00	0,00	0,00	0,00	0,00
c_3	0,00	0,00	0,00	0,00	0,00	0,67	0,00	0,83	0,83	0,83	0,83	0,83
c_4	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_5	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_6	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_7	0,00	0,00	0,67	0,17	0,00	0,00	0,00	0,00	0,11	0,11	0,11	0,11
c_8	0,00	0,00	0,67	0,17	0,00	0,00	0,00	0,11	0,00	0,11	0,11	0,11
c_9	0,00	0,00	0,67	0,17	0,00	0,00	0,00	0,11	0,11	0,00	0,11	0,11
c_{10}	0,00	0,00	0,67	0,17	0,00	0,00	0,00	0,11	0,11	0,11	0,00	0,11
c_{11}	0,00	0,00	0,67	0,17	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,00

Figura 40. DSM do Multi-MOM.

A solução de agrupamento que foi encontrada é descrita pela CMM da Figura 41 e pela DSM reordenada da Figura 42, na qual os valores de dependência externa aos módulos foram destacados em negrito.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
M_0	0	0	1	1	0	0	0	1	1	1	1	1
M_1	1	1	0	0	0	0	0	0	0	0	0	0
M_2	0	0	0	0	0	1	0	0	0	0	0	0
M_3	0	0	0	0	0	0	1	0	0	0	0	0
M_4	0	0	0	0	1	0	0	0	0	0	0	0

Figura 41. CMM da solução encontrada para o Multi-MOM.

A fim de avaliar a qualidade da solução encontrada, foi requisitado ao arquiteto do projeto que ele próprio gerasse uma solução de agrupamento. Para tanto, foi esclarecido o significado e o objetivo dessa atividade, mas não foram apresentadas métricas ou qualquer sugestão sobre como proceder, ou seja, a solução do arquiteto foi gerada com base no seu conhecimento sobre a arquitetura, de forma subjetiva.

$CTC \cong 8,65$													
		c_2	c_3	c_7	c_8	c_9	c_{10}	c_{11}	c_0	c_1	c_5	c_6	c_4
M_0	c_2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,00	0,67	0,00
	c_3	0,00	0,00	0,83	0,83	0,83	0,83	0,83	0,00	0,00	0,67	0,00	0,00
	c_7	0,67	0,17	0,00	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00
	c_8	0,67	0,17	0,11	0,00	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00
	c_9	0,67	0,17	0,11	0,11	0,00	0,11	0,11	0,00	0,00	0,00	0,00	0,00
	c_{10}	0,67	0,17	0,11	0,11	0,11	0,00	0,11	0,00	0,00	0,00	0,00	0,00
	c_{11}	0,67	0,17	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00
M_1	c_0	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,00	0,00
	c_1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,67	0,00	0,00	0,00	0,00
M_2	c_5	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M_3	c_6	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M_4	c_4	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Figura 42. DSM do Multi-MOM reordenada de acordo com a solução encontrada.

A solução do arquiteto é descrita pela CMM da Figura 43 e pela DSM reordenada da Figura 44.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
M_0	0	0	1	0	0	0	0	1	1	1	1	1
M_1	0	0	0	1	1	1	0	0	0	0	0	0
M_2	1	1	0	0	0	0	0	0	0	0	0	0
M_3	0	0	0	0	0	0	1	0	0	0	0	0

Figura 43. CMM da solução do arquiteto.

Pela solução do arquiteto, o agrupamento dos componentes *Service Manager* (c_0) e *Service Locator* (c_1) é confirmado como sendo adequado pela definição do módulo M_2 na Figura 44, bem como o agrupamento dos componentes *Message Dispatcher* (c_2), *Communication Paradigm - Publish/Subscribe* (c_7), *Communication Paradigm - Message Queue* (c_8), *Communication Paradigm - Tuple Space* (c_9), *Communication Paradigm - PTP Notification* (c_{10}) e *Communication Paradigm - Synchronous* (c_{11}) pela definição do módulo M_0 , e a separação do componente *Message Exchanger* (c_6) em um módulo individual.

Já os componentes *Message Manager Control* (c_3), *Persistence Manager* (c_4) e *TTL Monitor* (c_5), que constituem o componente composto *Message Manager*, e que na solução do arquiteto são agrupados juntos formando o módulo M_1 , foram mantidos separados na solução encontrada pelo algoritmo, o que na prática não constitui uma boa solução, já que geralmente componentes que constituem um componente composto são fortemente dependentes entre si.

$CTC \cong 9,44$													
		c_2	c_7	c_8	c_9	c_{10}	c_{11}	c_3	c_4	c_5	c_0	c_1	c_6
M_0	c_2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,67
	c_7	0,67	0,00	0,11	0,11	0,11	0,11	0,17	0,00	0,00	0,00	0,00	0,00
	c_8	0,67	0,11	0,00	0,11	0,11	0,11	0,17	0,00	0,00	0,00	0,00	0,00
	c_9	0,67	0,11	0,11	0,00	0,11	0,11	0,17	0,00	0,00	0,00	0,00	0,00
	c_{10}	0,67	0,11	0,11	0,11	0,00	0,11	0,17	0,00	0,00	0,00	0,00	0,00
	c_{11}	0,67	0,11	0,11	0,11	0,11	0,00	0,17	0,00	0,00	0,00	0,00	0,00
M_1	c_3	0,00	0,83	0,83	0,83	0,83	0,83	0,00	0,00	0,67	0,00	0,00	0,00
	c_4	0,00	0,00	0,00	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00
	c_5	0,00	0,00	0,00	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00
M_2	c_0	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00
	c_1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,67	0,00	0,00
M_3	c_6	0,00	0,00	0,00	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00

Figura 44. DSM do Multi-MOM reordenada de acordo com a solução do arquiteto.

A fim de compreender o resultado obtido pelo algoritmo, foi feita uma análise mais detalhada sobre a DSM do projeto, sendo identificado que esta não expressa uma forte dependência entre os componentes c_3 , c_4 e c_5 : $DSMc_3, c_4 = DSMc_4, c_5 = DSMc_5, c_4 = 0$, $DSMc_4, c_3 = DSMc_5, c_3 = 0,22$ e $DSMc_3, c_5 = 0,67$. Na solução encontrada pelo algoritmo, o componente c_3 é agrupado com o componente *Message Dispatcher* (c_2) e com os componentes variantes ($c_7 - c_{11}$), dado os elevados níveis de dependência entre c_3 e os componentes variantes: $DSMc_3, c_7 = DSMc_3, c_8 = DSMc_3, c_9 = DSMc_3, c_{10} = DSMc_3, c_{11} = 0,83$ e $DSMc_7, c_3 = DSMc_8, c_3 = DSMc_9, c_3 = DSMc_{10}, c_3 = DSMc_{11}, c_3 = 0,11$. Isso sugere que houve uma falha na avaliação da dependência entre os componentes na DSM.

Diante dessa questão, foi proposta a reconstrução da DSM do projeto. Para tanto, o projeto arquitetural descrito pela Figura 39 foi mantido, sendo feitas modificações no mapeamento das *features* para os componentes, conforme apresentado pela Tabela 12. De acordo com essa tabela, o componente *Message Manager Control* (c_3) passou a incluir as *features* *Persistência de mensagens* e *Monitor de TTL*, já que os componentes c_3 , *Persistence Manager* (c_4) e *TTL Monitor* (c_5) constituem o componente composto *Message Manager* e o componente c_3 é responsável por controlar a funcionalidade desse componente composto. Além disso, cada componente variante do tipo *Communication Paradigm* passou a incluir a *feature* *Paradigma de comunicação*, denotando que tais componentes estão relacionados à mesma funcionalidade geral.

Tabela 12. Relação de *features* por componente corrigida.

Índice	Componente	Feature	Quantidade
c_0	Service Manager	Localização de serviços	1
c_1	Service Locator	Localização de serviços	1
c_2	Message Dispatcher	Reflexão	1
c_3	Message Manager Control	Buffer de envio	3
		Persistência de mensagens	
		Monitor de TTL	
c_4	Persistence Manager	Persistência de mensagens	1
c_5	TTL Monitor	Monitor de TTL	1
c_6	Message Exchanger	Troca de mensagens pela rede	1
c_7	Communication Paradigm-Publish/Subscribe	Publish/Subscribe	2
		Paradigma de comunicação	
c_8	Communication Paradigm-Message Queue	Message Queue	2
		Paradigma de comunicação	
c_9	Communication Paradigm-Tuple Space	Tuple Space	2
		Paradigma de comunicação	
c_{10}	Communication Paradigm-PTP Notification	PTP Notification	2
		Paradigma de comunicação	
c_{11}	Communication Paradigm-Synchronous	Synchronous	2
		Paradigma de comunicação	

Com base na Tabela 12, a métrica DF foi recalculada para todos os componentes, sendo construída uma nova DSM para o projeto, descrita pela Figura 45.

$CTC \cong 12,67$												
	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
c_0	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_1	0,67	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_2	0,33	0,00	0,00	0,00	0,00	0,00	0,67	0,00	0,00	0,00	0,00	0,00
c_3	0,00	0,00	0,00	0,00	0,11	0,78	0,00	0,67	0,67	0,67	0,67	0,67
c_4	0,00	0,00	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_5	0,00	0,00	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_6	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c_7	0,00	0,00	0,67	0,00	0,00	0,00	0,00	0,00	0,11	0,11	0,11	0,11
c_8	0,00	0,00	0,67	0,00	0,00	0,00	0,00	0,11	0,00	0,11	0,11	0,11
c_9	0,00	0,00	0,67	0,00	0,00	0,00	0,00	0,11	0,11	0,00	0,11	0,11
c_{10}	0,00	0,00	0,67	0,00	0,00	0,00	0,00	0,11	0,11	0,11	0,00	0,11
c_{11}	0,00	0,00	0,67	0,00	0,00	0,00	0,00	0,11	0,11	0,11	0,11	0,00

Figura 45. DSM do Multi-MOM corrigida.

A partir dessa nova DSM, foi encontrada a solução descrita pela CMM da Figura 46 e pela DSM reordenada da Figura 47.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
M_0	0	0	1	0	0	0	0	1	1	1	1	1
M_1	0	0	0	1	1	1	1	0	0	0	0	0
M_2	1	1	0	0	0	0	0	0	0	0	0	0

Figura 46. CMM da nova solução encontrada para o Multi-MOM.

$CTC \cong 7,88$													
		c_2	c_7	c_8	c_9	c_{10}	c_{11}	c_3	c_4	c_5	c_6	c_0	c_1
M_0	c_2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,67	0,33	0,00
	c_7	0,67	0,00	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c_8	0,67	0,11	0,00	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c_9	0,67	0,11	0,11	0,00	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c_{10}	0,67	0,11	0,11	0,11	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c_{11}	0,67	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M_1	c_3	0,00	0,67	0,67	0,67	0,67	0,67	0,00	0,11	0,78	0,00	0,00	0,00
	c_4	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,00	0,00	0,00	0,00
	c_5	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,00	0,00	0,00	0,00
	c_6	0,00	0,00	0,00	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00
M_2	c_0	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33
	c_1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,67	0,00

Figura 47. DSM do Multi-MOM reordenada segundo a nova solução encontrada.

A solução do arquiteto de acordo com a DSM corrigida é descrita pela Figura 48.

$CTC \cong 7,9$													
		c_2	c_7	c_8	c_9	c_{10}	c_{11}	c_3	c_4	c_5	c_0	c_1	c_6
M_0	c_2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,67
	c_7	0,67	0,00	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c_8	0,67	0,11	0,00	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c_9	0,67	0,11	0,11	0,00	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c_{10}	0,67	0,11	0,11	0,11	0,00	0,11	0,00	0,00	0,00	0,00	0,00	0,00
	c_{11}	0,67	0,11	0,11	0,11	0,11	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M_1	c_3	0,00	0,67	0,67	0,67	0,67	0,67	0,00	0,11	0,78	0,00	0,00	0,00
	c_4	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,00	0,00	0,00	0,00
	c_5	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00	0,00	0,00	0,00	0,00
M_2	c_0	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,33	0,00
	c_1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,67	0,00	0,00
M_3	c_6	0,00	0,00	0,00	0,00	0,00	0,00	0,22	0,00	0,00	0,00	0,00	0,00

Figura 48. DSM do Multi-MOM reordenada de acordo com a solução do arquiteto.

Conforme pode ser observado, a nova solução mostra-se bem mais coerente com a solução definida pelo arquiteto, exceto pelo fato de que o componente *Message Exchanger* (c_6) também foi agrupado com os componentes *Message Manager Control* (c_3), *Persistence Manager* (c_4) e *TTL Monitor* (c_5), ao contrário da solução do arquiteto, segundo a qual o componente c_6 deve ficar isolado em um módulo. Dado que o componente c_6 possui relação de dependência apenas com os componentes *Message Dispatcher* (c_2) e c_3 , e que o módulo M_0 que contém c_2 é grande para ainda envolver c_6 , pelo valor de CTC foi percebido que era mais vantagem manter c_6 agrupado com c_3 , c_4 e c_5 do que mantê-lo isolado em um módulo individual.

Dado que o valor de CTC referente à solução do algoritmo é menor, é possível concluir que na perspectiva do objetivo de maximizar as dependências internas aos módulos e minimizar as dependências externas entre eles, a solução encontrada pelo algoritmo é melhor do que a solução apresentada pelo arquiteto do projeto. Isso pode ser confirmado comparando-se a MBM da nova solução encontrada pelo algoritmo (Figura 49a) e a MBM da solução do arquiteto (Figura 49b).

	M_0	M_1	M_2
M_0	1,00	0,12	0,06
M_1	0,60	0,32	0,00
M_2	0,00	0,00	0,18

(a) MBM da nova solução encontrada

	M_0	M_1	M_2	M_3
M_0	1,00	0,00	0,06	0,12
M_1	0,60	0,28	0,00	0,00
M_2	0,00	0,00	0,18	0,00
M_3	0,00	0,04	0,00	0,00

(b) MBM da solução do arquiteto

Figura 49. MBM das soluções definidas para o Multi-MOM.

Em ambas as MBM, as dependências externas aos módulos foram inferiores às dependências internas, exceto pela dependência entre os módulos M_0 e M_1 destacada em negrito, que é inferior à dependência interna ao módulo M_0 , mas superior à dependência interna à todos os demais módulos. Isso se deve à forte dependência entre os componentes variantes ($c_7 - c_{11}$) em M_0 e o componente c_3 em M_1 , conforme pode ser verificado na Figura 48. Além disso, em ambas as DSM o somatório das dependências externas aos módulos é inferior ao somatório das dependências internas, embora que a solução encontrada pelo algoritmo seja mais eficiente do que a solução sugerida pelo arquiteto: Na MBM da Figura 49a, o somatório das dependências internas é 1,5 e das dependências externas é 0,78, enquanto que na MBM da figura Figura 49b, o somatório das dependências internas é 1,46 e das externas, 0,82.

Dessa forma, o uso da abordagem mostra-se útil, uma vez que oferece uma solução alternativa melhor, atendendo ao seu objetivo fundamental de prover recomendações e assim, auxiliar na tomada de decisão. Além disso, a abordagem também se mostrou útil para identificar falhas de projeto, tal como no caso descrito, que resultou na reavaliação do mapeamento das *features* para os componentes.

6.4. Ginga-CC

O Ginga é a especificação de *middleware* para aplicações de TV digital do SBTVD (Sistema Brasileiro de TV Digital Terrestre) [Souza Filho et al. 2007] e sua arquitetura é composta por três módulos principais: (i) Ginga-NCL, que oferece um ambiente de execução para aplicações declarativas; (ii) Ginga-J, que oferece um ambiente de execução para aplicações Java; e (iii) Ginga *Common Core* (Ginga-CC), que oferece um conjunto de serviços de TV Digital comuns a ambos os ambientes de execução Ginga-NCL e Ginga-J, abstraindo o sistema operacional e o *hardware* subjacentes.

Inicialmente, o Ginga não foi projetado como uma linha de produtos. Entretanto, dada a heterogeneidade e as restrições de recursos das plataformas de execução, bem como os diferentes requisitos das aplicações que podem ser executadas sob tais plataformas, atualmente está sendo realizada a refatoração deste projeto para uma arquitetura de LPS, a fim de facilitar a instanciação de *middlewares* personalizados. Apesar disso, dado que o Ginga-CC deve ser reusado por todo *middleware* Ginga, o estudo de caso descrito neste trabalho foi desenvolvido apenas sobre a arquitetura deste módulo, considerando que esta consiste em uma arquitetura de domínio de uma LPS em que todos os componentes são obrigatórios.

A arquitetura do Ginga-CC é descrita pelo diagrama de componentes da Figura 50. Conforme pode ser observado, trata-se de um diagrama simplificado, no qual foram omitidos os relacionamentos entre o componente *Controller* (c_1) e todos os demais componentes, resultando na existência de componentes que aparecem desconectados (*ReturnChannel* (c_8) e *Security* (c_9)). De acordo com esse diagrama e com a Tabela 13, que identifica as *features* implementadas por cada componente, foram calculadas as métricas DI e DF para o projeto, resultando na construção da DSM apresentada pela Figura 51, na qual os valores diferentes de zero foram destacados em negrito para facilitar a visualização.

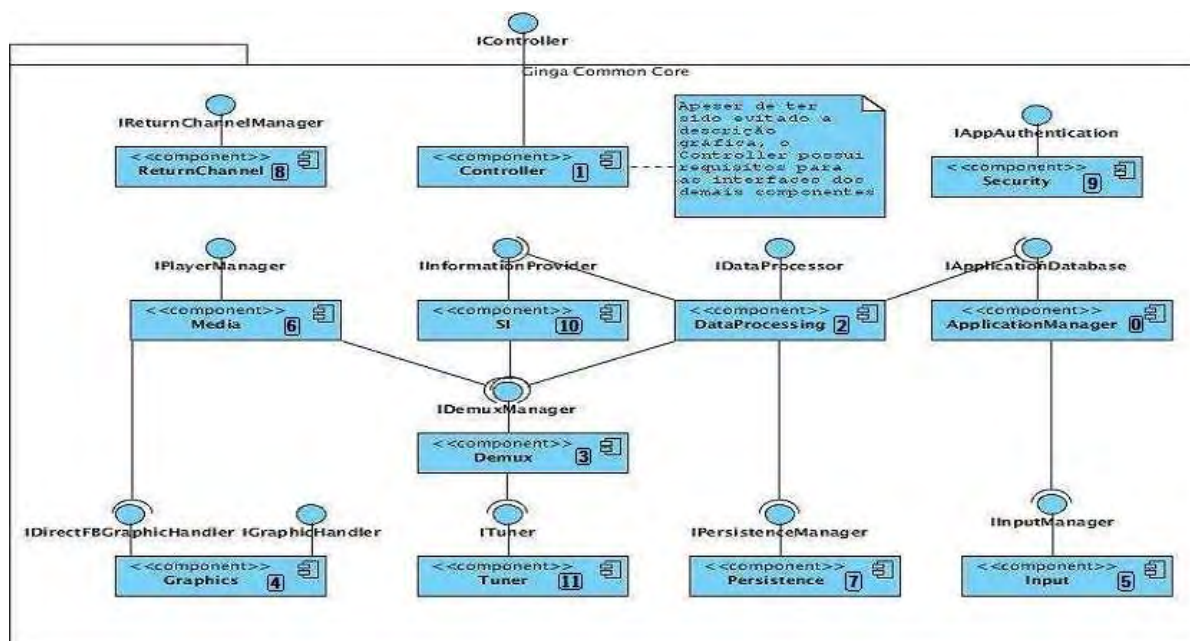


Figura 50. Diagrama de componentes do Ginga-CC [Souza Filho et al. 2007].

Tabela 13. Relação de *features* por componente do Ginga-CC.

Índice	Componente	Feature	Quantidade
0	ApplicationManager	NCL/Lua	4
		Java DTV	
		Resident	
		Access Control	
1	Controller	Access Interface	1
2	DataProcessing	Transport Stream	3
		Application Assembly	
		Application Persistence	
3	Demux	Demultiplexer Software	2
		Transport Stream	
4	Graphics	Audio	4
		Video	
		Text	
		Picture	
5	Input Manager	Input	1
6	Media	Audio	2
		Video	

7	Persistence	Application Persistence	1
8	ReturnChannel	Dial-Up	3
		Ethernet	
		ISDN	
9	Security	Java DTV	2
		Access Control	
10	Si	Transport Stream	1
11	Tuner	IP	2
		File System	

$CTC \cong 5,68$												
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}
C_0	0,00	0,06	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,00	0,00
C_1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
C_2	0,00	0,06	0,00	0,08	0,00	0,00	0,00	0,11	0,00	0,00	0,11	0,00
C_3	0,00	0,06	0,25	0,00	0,00	0,00	0,33	0,00	0,00	0,00	0,83	0,00
C_4	0,00	0,06	0,00	0,00	0,00	0,00	0,50	0,00	0,00	0,00	0,00	0,00
C_5	0,67	0,06	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
C_6	0,00	0,06	0,00	0,00	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00
C_7	0,00	0,06	0,28	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
C_8	0,00	0,06	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
C_9	0,17	0,06	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
C_{10}	0,00	0,06	0,28	0,17	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
C_{11}	0,00	0,06	0,00	0,67	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Figura 51. DSM do Ginga-CC.

A solução de agrupamento que foi encontrada pelo algoritmo, que é a solução ótima para o problema, é apresentada pela CMM da Figura 52 e pela DSM reordenada da Figura 53, na qual os valores de dependência externa aos módulos foram destacados em negrito.

	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}
M_0	0	0	1	1	0	0	0	0	0	0	1	1
M_1	1	0	0	0	0	1	0	0	0	1	0	0
M_2	0	1	0	0	0	0	0	1	1	0	0	0
M_3	0	0	0	0	1	0	1	0	0	0	0	0

Figura 52. CMM do Ginga-CC.

A fim de comprovar que a solução encontrada pelo algoritmo é realmente uma boa solução de agrupamento, foi realizada uma avaliação junto ao arquiteto do projeto, comparando-se a solução encontrada pelo algoritmo com a solução definida pelo arquiteto

segundo critérios definidos por ele próprio para caracterizar a dependência funcional entre os componentes.

$CTC \cong 2,64$													
		c_2	c_3	c_{10}	c_{11}	c_0	c_5	c_9	c_1	c_7	c_8	c_4	c_6
M_0	c_2	0,00	0,08	0,17	0,00	0,00	0,00	0,00	0,06	0,11	0,00	0,00	0,00
	c_3	0,25	0,00	0,83	0,00	0,00	0,00	0,00	0,06	0,00	0,00	0,00	0,33
	c_{10}	0,28	0,17	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00	0,00	0,00
	c_{11}	0,00	0,67	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00	0,00	0,00
M_1	c_0	0,17	0,00	0,00	0,00	0,00	0,00	0,17	0,06	0,00	0,00	0,00	0,00
	c_5	0,00	0,00	0,00	0,00	0,67	0,00	0,00	0,06	0,00	0,00	0,00	0,00
	c_9	0,00	0,00	0,00	0,00	0,17	0,00	0,00	0,06	0,00	0,00	0,00	0,00
M_2	c_1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	c_7	0,28	0,00	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00	0,00	0,00
	c_8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00	0,00	0,00
M_3	c_4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00	0,00	0,50
	c_6	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00	0,17	0,00

Figura 53. DSM do Ginga-CC reordenada de acordo com a solução encontrada.

A solução definida pelo arquiteto do projeto é descrita pela CMM da Figura 54 e pela DSM reordenada da Figura 55. De acordo com essa solução, o agrupamento dos componentes *Demux* (c_3), *Si* (c_{10}) e *Tuner* (c_{11}) foi mantido pela definição do módulo M_0 , bem como o agrupamento dos componentes *Application Manager* (c_0) e *Input Manager* (c_5), que compõem o módulo M_1 , e o agrupamento dos componentes *Graphics* (c_4) e *Media* (c_6) que definem o módulo M_3 .

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
M_0	0	0	0	1	0	0	0	0	0	0	1	1
M_1	1	0	0	0	0	1	0	0	0	0	0	0
M_2	0	0	1	0	0	0	0	1	0	0	0	0
M_3	0	0	0	0	1	0	1	0	0	0	0	0
M_4	0	1	0	0	0	0	0	0	0	0	0	0
M_5	0	0	0	0	0	0	0	0	1	0	0	0
M_6	0	0	0	0	0	0	0	0	0	1	0	0

Figura 54. CMM da solução definida pelo arquiteto.

O componente *Data Processing* (c_2), que na solução do arquiteto aparece agrupado com o componente *Persistence* (c_7), na solução encontrada pelo algoritmo (Figura 53) está inserido no módulo M_0 com os componentes *Demux* (c_3), *Si* (c_{10}) e *Tuner* (c_{11}). A razão disso deve-se ao fato das dependências entre o componente c_2 e os componentes c_3 e c_{10} serem superiores às dependências entre os componentes c_2 e c_7 : $DSMc_3, c_2 = 0,25$, $DSMc_{10}, c_2 =$

0,28, $DSM_{c_2, c_3} = 0,08$ e $DSM_{c_2, c_{10}} = 0,17$, enquanto que $DSM_{c_2, c_7} = 0,11$ e $DSM_{c_7, c_2} = 0,28$. Dessa forma, percebe-se que os valores de dependência não expressam a real dependência que provavelmente existe entre os componentes c_2 e c_7 , dada a solução do arquiteto, o que significa que uma reavaliação da documentação da arquitetura é recomendada, em particular do mapeamento entre *features* e componentes.

		$CTC \cong 2,99$											
		c_3	c_{10}	c_{11}	c_0	c_5	c_2	c_7	c_4	c_6	c_1	c_8	c_9
M_0	c_3	0,00	0,83	0,00	0,00	0,00	0,25	0,00	0,00	0,33	0,06	0,00	0,00
	c_{10}	0,17	0,00	0,00	0,00	0,00	0,28	0,00	0,00	0,00	0,06	0,00	0,00
	c_{11}	0,67	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00
M_1	c_0	0,00	0,00	0,00	0,00	0,00	0,17	0,00	0,00	0,00	0,06	0,00	0,17
	c_5	0,00	0,00	0,00	0,67	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00
M_2	c_2	0,08	0,17	0,00	0,00	0,00	0,00	0,11	0,00	0,00	0,06	0,00	0,00
	c_7	0,00	0,00	0,00	0,00	0,00	0,28	0,00	0,00	0,00	0,06	0,00	0,00
M_3	c_4	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,50	0,06	0,00	0,00
	c_6	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,17	0,00	0,06	0,00	0,00
M_4	c_1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
M_5	c_8	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00
M_6	c_9	0,00	0,00	0,00	0,17	0,00	0,00	0,00	0,00	0,00	0,06	0,00	0,00

Figura 55. DSM do Ginga-CC reordenada de acordo com a solução do arquiteto.

Situação semelhante acontece com os componentes *Security* (c_9), *Controller* (c_1) e *Return Channel* (c_8). O componente *Security* (c_9), que na solução do arquiteto aparece em um módulo isolado, na solução encontrada pelo algoritmo está agrupado com os componentes *Application Manager* (c_0) e *Input Manager* (c_5). Dado que o componente c_9 só possui relação de dependência com os componentes c_0 e c_1 , prevalecendo a dependência em relação à c_0 , e que o módulo que contém c_0 não é grande, em termos de custo faz mais sentido manter os componentes c_0 , c_5 e c_9 agrupados. Já os componentes *Controller* (c_1) e *Return Channel* (c_8), que na solução do arquiteto também aparecem em módulos isolados, na solução encontrada pelo algoritmo são agrupados juntos, com o acréscimo do componente *Persistence* (c_7), o que também faz sentido, dado que c_1 depende de c_7 e c_8 .

Novamente, em ambas as situações descritas, os valores de dependência não correspondem à percepção do arquiteto sobre a relação de dependência entre os componentes. Dado que a documentação do projeto não é detalhada o suficiente e que os resultados obtidos nos demais estudos de caso foram coerentes, são identificadas duas possíveis explicações para o ocorrido: falha/omissão na documentação do projeto ou falha/omissão na avaliação de

dependência por parte do arquiteto. Tal conclusão ainda é reforçada pelo fato do algoritmo se mostrar coerente ao propósito de maximizar dependências internas aos módulos e minimizar dependências entre módulos, resultando em um valor de CTC inferior. Isso é refletido na MBM dessas soluções, conforme pode ser verificado pela Figura 56. Em ambas as matrizes, as dependências externas aos módulos foram destacadas em negrito.

	M_0	M_1	M_2	M_3
M_0	1,00	0,00	0,14	0,13
M_1	0,07	0,41	0,07	0,00
M_2	0,11	0,00	0,05	0,00
M_3	0,00	0,00	0,05	0,27

(a) MBM da solução encontrada pelo algoritmo

	M_0	M_1	M_2	M_3	M_4	M_5	M_6
M_0	1,00	0,32	0,20	0,00	0,00	0,00	0,11
M_1	0,15	0,23	0,00	0,00	0,00	0,00	0,07
M_2	0,00	0,00	0,40	0,00	0,00	0,00	0,07
M_3	0,00	0,10	0,00	0,40	0,00	0,10	0,07
M_4	0,00	0,00	0,00	0,00	0,00	0,00	0,04
M_5	0,00	0,00	0,00	0,10	0,00	0,00	0,04
M_6	0,00	0,00	0,00	0,00	0,00	0,00	0,00

(b) MBM da solução do arquiteto

Figura 56. MBM das soluções definidas para o Ginga-CC.

A MBM da Figura 56a, que está associada à solução encontrada pelo algoritmo, no geral apresenta valores de dependência externa aos módulos inferiores à dependência interna deles, exceto para o módulo M_2 , que possui baixa dependência interna. Por sua vez, a MBM da Figura 56b possui três módulos com dependência interna nula, dado que são módulos unitários, resultando no aumento da dependência entre módulos. Isso pode ser comprovado comparando o somatório das dependências internas e externas para cada MBM. Na MBM da Figura 56a, o somatório das dependências internas aos módulos é 1,73 e o somatório das dependências externas entre os módulos é 0,52, enquanto que na MBM da Figura 56b, o somatório das dependências internas é 2,03 e das dependências externas é 1,37.

6.5. Considerações Finais

Em todos os estudos de caso apresentados, foram realizadas 50 execuções do algoritmo. Os dados relevantes para a análise do comportamento do algoritmo na perspectiva de otimização são resumidos na Tabela 14, segundo o cálculo da média entre as várias execuções. São eles: o custo da solução retornada, o número de soluções que foram avaliadas até se chegar à solução retornada (campo *tentativa* na tabela), a quantidade de vezes em que a temperatura foi atualizada, a quantidade de soluções de custo superior que foram aceitas, o total de soluções avaliadas e o tempo de processamento.

Tabela 14. Resultados da execução do algoritmo.

	Estudo de caso	Custo	Tentativa	Temperaturas usadas	Soluções ruins aceitas	Soluções visitadas	Tempo (s)
1	Microondas (DI e DF)	22,56	59.357	748	62.538	310.800	232,76
2	Microondas (DO e DF)	22,88	65.519	1.058	86.515	444.304	375,2
3	E-Commerce (iteração 1)	22,57	18.350	724	6.381	181.230	92,84
4	E-Commerce (iteração 2)	32,2	89.941	1.186	33.897	616.824	645,5
5	Multi-MOM	7,88	14.950	702	13.001	168.480	82
6	Ginga-CC	2,64	10.026	411	8.977	102.895	46,08

De acordo com essa tabela e com as soluções reportadas para cada estudo, observa-se que nas execuções de cada estudo de caso, o algoritmo resultou na mesma solução, com exceção da segunda iteração do estudo de caso da LPS de E-Commerce (caso 2), para o qual houve uma pequena variação entre as soluções geradas, traduzida pela média apresentada na tabela, cujo valor é ligeiramente superior ao valor reportado como solução na seção 6.2.2 (a solução reportada é a melhor dentre as soluções encontradas).

De acordo com essa tabela, também é possível observar que o tempo de execução do algoritmo é reduzido, tendo seu valor máximo para a segunda iteração do estudo de caso da LPS de E-Commerce (caso 2), o que é justificável, dado que o espaço de busca deste caso é o maior dentre os demais, o que leva a uma maior diversidade de soluções, fazendo a busca demorar mais para estabilizar-se.

Além disso, dado o número de soluções avaliadas e o índice da solução retornada (campo *tentativa* na Tabela 14), observa-se que a melhor solução avaliada geralmente é encontrada no “início” da busca, onde a probabilidade de aceitação de soluções de custo superior tende a ser maior. Esse fenômeno reforça o conceito de que ao aceitar soluções piores, a metaheurística de SA promove uma melhor exploração do espaço de busca, fazendo com que boas soluções sejam encontradas mais rapidamente.

Por fim, observa-se que a quantidade de soluções de custo superior aceita é pequena para a busca como um todo, dado o número de soluções avaliadas, o que também é coerente, já que a probabilidade de aceitação de soluções de custo superior é decrescida ao longo da busca até chegar ao ponto de anular-se.

Quanto à qualidade das soluções encontradas pelo algoritmo, com base na descrição detalhada dos estudos de caso, percebe-se que, no geral, as soluções são adequadas na perspectiva de maximizar as dependências internas aos módulos e minimizar as dependências externas entre eles. No entanto, para alguns problemas nem sempre é possível chegar a uma solução que mantém essa característica para todos os módulos. Esse é o caso do Ginga-CC (estudo de caso 6 na Tabela 14), por exemplo, onde existe um componente central na arquitetura que mantém relação de dependência com todos os demais componentes, levando à uma solução de agrupamento em que o módulo ao qual pertence o componente central possui dependência externa (ligeiramente) superior à dependência interna.

Além disso, também foi observado que as soluções encontradas definem módulos que não seriam dedutíveis pelo diagrama de componentes simplesmente. Isso sugere que o uso da métrica DF complementa as métricas DI e DO, e que o uso da abordagem como um todo é realmente válido, atendendo ao seu objetivo fundamental de prover recomendações e assim, auxiliar na tomada de decisão sobre como alocar equipes de desenvolvimento aos componentes em um projeto. Lembrando que a decisão em si depende da avaliação de outros critérios, tais como a complexidade dos componentes/módulos a serem desenvolvidos, a habilidade técnica das equipes disponíveis para desenvolvê-los, a facilidade de interação entre as equipes, além de aspectos relacionados ao custo e ao tempo de desenvolvimento. Logo, a presente abordagem cobre apenas um dos aspectos envolvidos nessa decisão, que é a organização dos componentes em módulos visando minimizar a necessidade de comunicação entre equipes.

Apesar disso, para os estudos de caso do Multi-MOM (caso 5) e do Ginga-CC (caso 6), para os quais foi possível comparar a solução encontrada pelo algoritmo com a solução sugerida pelo arquiteto do projeto, observou-se que alguns valores de dependência entre componentes não corresponderam à percepção do arquiteto sobre tais relações. Dado que a documentação de ambos os projetos não é rica, inclusive tendo sido identificada uma falha na documentação do Multi-MOM, e que os resultados obtidos no geral foram coerentes com o propósito de maximizar dependências internas aos módulos e minimizar as dependências externas a estes, é sugerida uma reavaliação da documentação desses projetos, não podendo ser excluída a possibilidade de falhas ou omissões na avaliação de dependência por parte do arquiteto.

O uso das métricas requer uma documentação detalhada, que tenha sido atualizada durante o projeto arquitetural para garantir que esteja coerente. Falhas ou omissões na documentação resultam em valores de dependência incoerentes, levando o algoritmo à resultados de agrupamento incompatíveis ao entendimento do gerente de projeto sobre a arquitetura. Por sua vez, a avaliação do arquiteto é subjetiva e não é apoiada por nenhuma ferramenta, sendo natural que ocorram omissões ou falhas nesse processo. De toda forma, é necessário que a presente abordagem seja avaliada em mais estudos de caso, e que inclusive sejam feitos testes com outros valores do parâmetro α nas Equações (12) e (13) utilizadas para reduzir as métricas DI, DO e DF à um valor único, a fim de avaliar tais métricas de forma mais completa.

Capítulo 7

Considerações Finais

Neste trabalho, foi apresentada uma abordagem baseada em evidência quantitativa para o agrupamento de componentes de software fortemente dependentes em módulos no contexto de projetos distribuídos de LPS. A idéia é que módulos sejam fracamente dependentes entre si a fim de poderem ser implementados por equipes de desenvolvimento distintas com reduzida necessidade de comunicação e coordenação. A abordagem consiste na primeira etapa de um *framework* de recomendação para alocação de equipes em projetos distribuídos de LPS, sendo responsável por definir os módulos e prover informação sobre o nível de dependência entre eles, informação esta que é usada para inferir sobre a necessidade de comunicação entre as equipes que irão desenvolvê-los.

Diversas abordagens têm sido propostas para agrupar componentes de software. Entretanto, tais abordagens apresentam severas limitações para serem aplicadas ao propósito deste trabalho: (i) geralmente lidam com dependência em nível de código de implementação, não sendo adequadas para serem utilizadas para fins de alocação; (ii) não discutem em profundidade como deve ser feita a avaliação da dependência, seja tratando-a como uma atividade subjetiva, que deve ser realizada pelo desenvolvedor segundo seus próprios critérios, seja limitando-a a uma simples contagem, não oferecendo condições de fazer uma avaliação comparativa adequada entre os relacionamentos de dependência examinados; (iii) a maioria das abordagens não lida com LPS, cujos aspectos de opcionalidade e variabilidade requerem tratamento diferenciado; (iv) utilizam estratégias de agrupamento ineficientes, cujas deficiências já são bem conhecidas na literatura (e discutidas no Capítulo 3); e por fim, (v) as abordagens não tratam da dependência entre os módulos definidos a partir do agrupamento dos componentes, informação esta necessária para a alocação de equipes aos módulos.

A abordagem desenvolvida, por sua vez, oferece: (i) medidas quantitativas para caracterizar dependências em nível arquitetural entre componentes de software; (ii) orientação

sobre como obter tais medidas no contexto de LPS; (iii) um algoritmo de agrupamento baseado em SA para recomendar módulos fracamente dependentes entre si; e (iv) medida quantitativa para caracterizar dependências entre módulos.

Dessa forma, a abordagem supera as limitações das abordagens existentes e apresenta uma nova visão sobre o problema de agrupamento de componentes: (i) a solução do problema contribui para a construção de metodologias para lidar com o desafio da comunicação no contexto de desenvolvimento distribuído de software, tal como o *framework* de recomendação de alocação; (ii) a arquitetura de software é utilizada como fonte central de informação para estruturação do problema, o que por consequência, enaltece o papel da arquitetura como ferramenta de coordenação; por fim, (iii) o problema é estruturado como um problema de otimização, ao qual é aplicado um algoritmo baseado em metaheurística, contribuindo com a consolidação da pesquisa numa nova vertente da Engenharia de Software denominada Engenharia de Software Baseada em Busca (*Search Based Software Engineering*).

A fim de demonstrar o uso da abordagem e sua eficácia, foram desenvolvidos quatro estudos de caso, os quais foram detalhadamente documentados. Tais estudos demonstraram que os resultados alcançados atendem ao objetivo de agrupar componentes fortemente dependentes entre si, significando que as métricas são adequadas para caracterizar a dependência entre componentes e que o algoritmo de agrupamento proposto é eficaz.

7.1. Contribuições

Como contribuição inicial, a abordagem desenvolvida define métricas arquiteturais que visam mensurar o nível de dependência entre pares de componentes, tornando essa atividade objetiva e simples, sendo também oferecidas orientações sobre como lidar com os mecanismos de variabilidade de um projeto arquitetural. As métricas estão baseadas nas interfaces e operações providas e requeridas pelos componentes, e nas *features* por eles implementadas, as quais retratam relacionamentos nem sempre perceptíveis pelas interfaces e/ou operações. Lembrando que como é possível utilizar as métricas de forma independente e a definição de interfaces/operações providas e requeridas é inerente ao desenvolvimento baseado em componentes, utilizando-se apenas a métrica baseada em interfaces ou a métrica baseada em operações, é possível aplicar a abordagem em projetos de DBC que não sejam de LPS, ampliando o uso da abordagem para outros cenários.

Além disso, como contribuição, a abordagem define um algoritmo baseado em *Simulated Annealing* para encontrar uma solução de agrupamento de componentes que seja

satisfatória, ou seja, que se aproxime (e quando possível alcance) a solução ótima sem despendar um elevado custo computacional. O algoritmo desenvolvido difere dos algoritmos de SA identificados na literatura para esse fim por: utilizar uma equação de resfriamento de melhor desempenho (soluções com qualidade superior encontradas em tempo reduzido); definir uma função objetivo que avalia a qualidade de uma solução considerando as dependências entre componentes e o tamanho dos agrupamentos, não sendo necessário definir o tamanho desejado para os agrupamentos, nem a quantidade destes; e aceitar como entrada uma solução previamente definida, lidando com projetos desenvolvidos em múltiplas iterações.

Como contribuição final, visando auxiliar na alocação de equipes no contexto de DDS, a abordagem ainda oferece uma medida quantitativa para avaliar a dependência entre módulos de software. Para viabilizar tais contribuições, a abordagem estrutura o problema de agrupamento de componentes de software em uma solução com três etapas com entradas e saídas bem definidas e objetivos bem delimitados, de maneira que cada uma delas pode ser trabalhada de forma relativamente independente. Isto significa que novas métricas arquiteturais podem ser introduzidas, por exemplo, assim como novos algoritmos de agrupamento podem ser propostos ou mesmo podem ser feitos ajustes no algoritmo desenvolvido, como substituir a equação que calcula a probabilidade de uma solução inferior. Em outras palavras, a abordagem desenvolvida é flexível, estando aberta a adaptações e extensões.

7.2. Limitações

Apesar das relevantes contribuições, a abordagem desenvolvida apresenta algumas limitações. Primeiramente, para que a abordagem seja aplicável, é necessário que a arquitetura do projeto esteja bem definida, estável e bem documentada. Dessa forma, o presente trabalho está baseado no desenvolvimento de linhas de produto segundo um processo proativo. Entretanto, se for admitida uma linha de produto desenvolvida segundo um processo extrativo (engenharia reversa), a abordagem já não se mostra adequada. Uma possibilidade nesse caso seria definir métricas que pudessem ser aplicadas em nível de código. No entanto, uma vez que os componentes já estariam implementados, não faria sentido aplicar a abordagem visando dar suporte à alocação de equipes de desenvolvimento. Nessa situação, a abordagem seria útil para fazer uma avaliação comparativa entre o projeto arquitetural e o projeto real, a

fim de avaliar a qualidade da arquitetura, tal como outras abordagens existentes na literatura, fugindo de seu foco original.

A abordagem também não lida com o aspecto de complexidade de desenvolvimento dos módulos (ou dificuldade de desenvolvimento destes) segundo o conhecimento requerido para implementar cada um de seus componentes, fator relevante no processo de alocação de equipes de desenvolvimento aos módulos. Nesta abordagem, a complexidade é considerada apenas na perspectiva da quantidade de componentes em cada módulo.

Outra limitação do trabalho diz respeito à sua validação. A avaliação completa da abordagem desenvolvida não é trivial, já que requer a avaliação conjunta das métricas que caracterizam a dependência entre os componentes e entre módulos, e do algoritmo de agrupamento. Durante o desenvolvimento do trabalho, as métricas foram fixadas e avaliações experimentais foram feitas para ajustar os parâmetros do algoritmo, sendo possível confirmar a coerência das métricas e do algoritmo pelos resultados dos estudos de caso. Assim, avaliações mais amplas devem ser feitas com projetos de complexidade e tamanho variados, a fim de efetivamente validar a abordagem. Inclusive, a comparação completa entre este trabalho e os trabalhos relacionados (em engenharia de software) não foi possível, por causa da diferença entre a natureza das informações usadas e dos propósitos envolvidos, por exemplo: dependência em nível de código versus dependência em nível arquitetural; função CTC versus função MQ; metaheurística *Simulated Annealing* versus *Hill Climbing*. A abordagem foi projetada visando superar as limitações identificadas nas abordagens existentes, mas falta um estudo comparativo amplo para certificar que esse propósito foi alcançado, dadas as múltiplas variáveis envolvidas.

Por fim, o algoritmo para agrupamento de componentes foi implementado, mas não foi desenvolvida uma interface gráfica para facilitar a introdução dos dados de entrada e para apresentar os resultados obtidos. Dessa forma, até o momento, o cálculo das métricas é feito manualmente ou através de algum programa que facilita a criação de planilhas, e os resultados são apresentados em arquivos de *log*.

7.3. Trabalhos Futuros

Dadas as limitações identificadas, como trabalho futuro primeiramente é planejado o desenvolvimento de uma ferramenta para dar suporte ao uso da presente abordagem, que deve oferecer recurso gráfico para facilitar a manipulação dos dados e a apresentação dos resultados. A ferramenta também deve permitir que o usuário crie suas próprias soluções, bem

como faça modificações nas soluções geradas pelo algoritmo quando julgar necessário, sendo informado sobre a qualidade das soluções construídas e das modificações realizadas.

Em seguida, é planejada a realização de mais estudos de caso, com projetos de maior complexidade e tamanho do que aqueles apresentados nesta dissertação, a fim de realizar uma avaliação mais completa da abordagem desenvolvida. Uma avaliação comparativa detalhada entre o presente trabalho e os trabalhos relacionados discutidos no Capítulo 3 também é pretendida, a fim de complementar a avaliação da abordagem.

Por fim, pretende-se adicionar métricas à abordagem para avaliar a complexidade de desenvolvimento de componentes e de módulos. Para tanto, se faz necessário realizar uma revisão na literatura para verificar a existência de métricas para esse fim ou de métricas que possam ser adaptadas para esse propósito, considerando que as métricas devem ser aplicadas em nível arquitetural.

Referências

- [Abramowitz, Stegun 1972] ABRAMOWITZ, M., STEGUN, I. A. *Stirling Numbers of the Second Kind*. 24.1.4 in Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing. New York, Dover, pp. 824-825.
- [Albin 2003] ALBIN, S. T. *The Art of Software Architecture: Design Methods and Techniques*. John Wiley & Sons.
- [Allen 1997] ALLEN, T. J. *Architecture and Communication Among Product Development Engineers*. Sloan School of Management, MIT, Sloan Working Paper, #3983.
- [Blois et al. 2005] BLOIS, A.P., WERNER, C., BECKER, K. *Towards a Components Grouping Technique within a Domain Engineering Process*. Proceedings of the 31th Euromicro Conference, Component-based Software Engineering Track, Porto, Portugal, pp.18-25.
- [Bosch 2001] BOSCH, J. *Software product lines: organizational alternatives*. Proceedings of the 23rd international Conference on Software Engineering. IEEE Computer Society, Washington, DC, pp. 91-100.
- [Brownsword, Clements 1996] BROWNSWORD, L., CLEMENTS, P. *A Case Study in Successful Product Line Development*. Technical Report CMU/SEI-96-TR-016, Software Engineering Institute, Carnegie Mellon University.
- [Carmel 1999] CARMEL, E. 1999. *Global Software Teams – Collaborating Across Borders and Time-Zone*. Prentice Hall.
- [Conway 1968] CONWAY, M. E. *How do committees invent?* Datamation. Vol. 14,4. pp. 28–31.
- [Costa et al. 2008] COSTA, J. M. R, FEITOSA, R. M., DE SOUZA, C. R. B. *RaisAware: Uma Ferramenta de Auxílio À Engenharia de Software Colaborativa Baseada em Análises de Dependências*. Proceedings of the Simpósio Brasileiro de Sistemas Colaborativos (SBSC), pp. 254-264.
- [Czarnecki, Eisenecker 2000] CZARNECKI, K., EISENECKER, U. W. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co.
- [Czarnecki et al. 2005] CZARNECKI, K., HELSEN, S., EISENECKER, U. *Formalizing Cardinality-based Feature Models and their Specialization*. Software Process: Improvement and Practice. Vol. 10, 1. pp. 7-29.
- [Davis 1991] DAVIS, L. *Handbook of Genetic Algorithms*. New York, Van Nostrand Reinhold.
-

- [De Souza et al. 2004] DE SOUZA, C. R. B., REDMILES, D., CHENG, L., MILLEN, D., and PATTERSON, J. *Sometimes you need to see through walls: a field study of application programming interfaces*. Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, Chicago, IL, ACM Press, pp. 63-71.
- [Dijkstra, 1976] DIJKSTRA, EDSGER W. *A Discipline of Programming*. Prentice Hall.
- [Doval et al. 1999] DOVAL, D., MANCORIDIS, S., MITCHELL, B. S. *Automatic clustering of software systems using a genetic algorithm*. Proceedings of the International Conference on Software Tools and Engineering Practice. IEEE Computer Society, Washington, DC, USA, pp. 73.
- [Espinosa et al. 2007] ESPINOSA, J. A., SLAUGHTER, S. A., KRAUT, R. E., HERBSLEB, J. D. *Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development*. Organization Science. Vol. 18, 4. pp. 613-630.
- [Etzeberria, Sagardui 2005] ETXEVERRIA, L., SAGARDUI, G. *Product-line architecture: New issues for evaluation. Lecture Notes in Computer Science*, ISBN 3-540-28936-4, Springer-Verlag GmbH. Vol. 3714, pp. 174-185.
- [Fernandez 1998] FERNANDEZ, C. I. G. *Integration analysis of product architecture to support effective team co-location*, Master Thesis, Massachusetts Institute of Technology.
- [Ghezzi et al. 2003] GHEZZI, C., JAZAYERI, M., MANDRIOLI, D. *Fundamentals of Software Engineering*, Upper Saddle River, NJ, Prentice Hall.
- [Gill, Balkishan 2008] GILL, N. S., and BALKISHAN. *Dependency and interaction oriented complexity metrics of component-based systems*. SIGSOFT Software Engineering Notes. Vol. 33, 2. pp. 1-5.
- [Gomaa 2004] GOMAA, H. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Object-Oriented Technology Series, ISBN: 0201775956.
- [Gomaa et al. 1992] GOMAA, H., KERSCHBERG, L., SUGUMARAN, V. *A Knowledge-Based Approach to Generating Target System Specifications from a Domain Model*. In Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, North-Holland Publishing Co., Amsterdam, The Netherlands. Vol. 1, pp. 252-258.
- [Grinter 2003] GRINTER, R. E. *Recomposition: Coordinating a Web of Software Dependencies*. Computer Supported Cooperative Work. Vol. 12, 3. pp. 297-327.
- [Grinter et al. 1999] GRINTER, R. E., HERBSLEB, J. D., PERRY, D. E. *The Geography of Coordination: Dealing with Distance in R&D Work*. In Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, ACM Press, New York, pp. 306-315.
- [Griss et al. 1998] GRISS, M., FAVARO, J., D'ALESSANDRO, M. *Integrating Feature Modeling with RSEB*. In Proceedings of the 5th International Conference on Software Reuse. IEEE Computer Society. Victoria, Canadá. pp. 76-85.
-

- [Harman, Jones 2001] HARMAN, M., JONES, B. F. *Search-based software engineering*. Information & Software Technology. Vol. 43, 14. pp. 833-839.
- [Harman 2007] HARMAN, M. *The Current State and Future of Search Based Software Engineering*. In Proceedings of the 2007 Future of Software Engineering (FOSE '07). IEEE Computer Society, Washington, DC, USA, pp. 342-357.
- [Harman et al. 2002] HARMAN, M., HIERONS, R., PROCTOR, M. *A new representation and crossover operator for search-based optimization of software modularization*. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1351-1358.
- [Harman et al. 2005] HARMAN, M., SWIFT, S., MAHDAVI, K. *An empirical study of the robustness of two module clustering fitness functions*. In Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO '05), Hans-Georg Beyer (Ed.). ACM, New York, NY, USA, pp. 1029-1036.
- [Helmer et al. 2008] HELMER, R. et al. *Systematic Module and Interface Definition Using component DSM*. Engineering Design, DOI: 10.1080/09544820802563226.
- [Herbsleb, Grinter 1999] HERBSLEB, J. D., GRINTER, R. E. *Architectures, Coordination, and Distance: Conway's Law and Beyond*. IEEE Software. Vol. 16, 5. pp. 63-70.
- [Herbsleb 2007] HERBSLEB, J. D. 2007. *Global Software Engineering: The Future of Socio-technical Coordination*. In Proceedings of the 2007 Future of Software Engineering (FOSE '07). IEEE Computer Society, Washington, DC, USA, pp. 188-198.
- [Hoek et al. 2003] HOEK, A. V. D., DINCEL, E., MEDVIDOVIC, N. *Using Service Utilization Metrics to Assess the Structure of Product Line Architectures*. In Proceedings of the 9th International Software Metrics Symposium (Metrics'03). IEEE Computer Society, Washington, DC, USA, Press. pp. 298-308.
- [Idicula 1995] IDICULA, J. *Planning for Concurrent Engineering*. Master Thesis, Nanyang Technological University.
- [IEEE 2000] IEEE Std. 1471-2000. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*.
- [Jaccard 1901] JACCARD, P. *Étude comparative de la distribution florale dans une portion des Alpes et des Jura*, Bulletin de la Société Vaudoise des Sciences Naturelles. Vol. 37, pp. 547-579.
- [Kirkpatrick et al. 1983] KIRKPATRICK, S.; GELATT, C. D. JR.; VECCHI, M. P. *Optimization by Simulated Annealing*. Science. Vol. 220, 4598. pp. 671-680.
- [Kobitzsch et al. 2001] KOBITZSCH, W., ROMBACH, D., FELDMANN, R. L. *Outsourcing in India*. IEEE Software. Vol. 18, 2. pp. 78-86. DOI= <http://dx.doi.org/10.1109/52.914751>.
- [Lamersdorf, Munch 2009] LAMERSDORF, A., MUNCH, J. *TAMRI: A Tool for Supporting Task Distribution in Global Software Development Projects*. In Proceedings of the 4th IEEE international Conference on Global Software Engineering, pp. 322-327.
-

- [Linden et al. 2007] LINDEN, F. J., SCHMID, K., ROMMES, E. *Software Product Lines in Action: the Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc.
- [Lopes, Bajracharya 2005] LOPES, C. V. and BAJRACHARYA, S. K. *An analysis of modularity in aspect oriented design*. In Proceedings of the 4th international Conference on Aspect-Oriented Software Development (AOSD '05). ACM, NY, pp. 15-26.
- [MacCormack et al. 2008] MACCORMACK, A. D., RUSNAK, J., BALDWIN, C. Y. *Exploring the Duality between Product and Organizational Architectures: A Test of the Mirroring Hypothesis*. Harvard Business School Working Papers 08-039, Harvard Business School.
- [Mahdavi et al. 2003] MAHDAVI, K., HARMAN, M., HIERONS, R. M. *A multiple hill climbing approach to software module clustering*. In Proceedings of the IEEE International Conference on Software Maintenance (ICSM '03). IEEE Computer Society, Washington, DC, USA, pp. 315-324.
- [Mancoridis et al. 1998] MANCORIDIS, S., et al. *Using Automatic Clustering to Produce High-Level System Organizations of Source Code*. In Proceedings of the 6th International Workshop on Program Comprehension (IWPC '98). IEEE Computer Society, Washington, DC, USA, pp. 45-53.
- [Mancoridis et al. 1999] MANCORIDIS, S., et al. *Bunch: A clustering tool for the recovery and maintenance of software system structures*. In Proceedings of the IEEE International Conference on Software Maintenance (ICSM '99). IEEE Computer Society, Washington, DC, USA, pp. 50-59.
- [Metropolis et al. 1953] METROPOLIS, N.; ROSENBLUTH, A.; ROSENBLUTH, M.; TELLER, A. *Equations of State Calculations by Fast Computing Machines*. Chemical Physics. Vol. 21, 6. pp. 1087-1092.
- [Miler 2000] MILER JÚNIOR, N. *A Engenharia de Aplicações no Contexto da Reutilização Baseada em Modelos de Domínio*. Dissertação de Mestrado. UFRJ/COPPE.
- [Mitchell, Mancoridis 2002] MITCHELL, B. S., MANCORIDIS, S. *Using Heuristic Search Techniques to Extract Design Abstractions from Source Code*. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02), pp. 1375-1382.
- [Mockus, Weiss 2001] MOCKUS, A., WEISS, D. M. *Globalization by chunking: a quantitative approach*. IEEE Software. Vol. 18, 2. pp. 30-37, DOI: 10.1109/52.914737.
- [Morais, Elias 2010] MORAIS, Y., ELIAS, G. *Integrating Communication Paradigms in a Mobile Middleware Product Line*. In Proceedings of the 9th International Conference on Networks (ICN '10). IEEE Computer Society, Washington, DC, USA, pp. 255-261.
- [Oliveira Junior et al. 2007] OLIVEIRA JUNIOR, E. A., MALDONADO, J. C., GIMENES, I. M. S. *Uma Revisão Sistemática sobre Avaliação de Linha de Produto de Software*. Relatório técnico. Instituto de Ciências Matemáticas e de Computação (ICMC). ISSN - 0103-2569.
-

- [Ovaska et al. 2003] OVASKA, P., ROSSI, M., MARTTIIN, P. *Architecture as a Coordination Tool in Multi-site Software Development*. Software Process: Improvement and Practice. Vol. 8, 4. pp. 233-247.
- [Parnas 1972] PARNAS, D. L. *On the criteria to be used in decomposing systems into modules*. Communications of the ACM. Vol. 15, 12. pp. 1053-1058.
- [Paulish 2003] PAULISH, D.J. *Product Line Engineering for Global Development*. In Proceedings of the 3rd International Workshop on Product Line Engineering The Early Steps: Planning, Modeling, and Managing (PLEES '03), pp. 17-22.
- [Pereira et al. 2010] PEREIRA, T. A. B., SANTOS, V. S., RIBEIRO, B. L., ELIAS, G. A. *A Recommendation Framework for Allocating Global Software Teams in Software Product Line Projects*. In Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE '10). ACM Press, New York, pp. 36-40.
- [Pohl et al. 2005] POHL, K., BCKLE, G., VAN DER LINDEN, F. J. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, New York, USA.
- [Praditwong et al. 2010] PRADITWONG, K., HARMAN, M., YAO, X. *Software Module Clustering as a Multi-Objective Search Problem*. IEEE Transactions on Software Engineering. Vol. 37, 2. pp. 264-282.
- [Repenning et al. 2001] REPENNING, A., IOANNIDOU, A., PAYTON, M., YE, W., and ROSCHELLE, J. *Using Components for Rapid Distributed Software Development*. IEEE Software. Vol. 18, 2. pp. 38-45. DOI= <http://dx.doi.org/10.1109/52.914739>.
- [Rota 1964] ROTA, G. C. *The American Mathematical Monthly*, Vol. 71, 5. pp. 498-504.
- [Roy, Graham 2008] ROY, B., GRAHAM, T. C. N. *Methods for Evaluating Software Architecture: A Survey*. Technical Report 2008-545. School of Computing, Queen's University at Kingston. Ontario, Canadá.
- [Sangal et. al 2005] SANGAL, N., JORDAN, E., SINHA, V., JACKSON, D. *Using dependency models to manage complex software architecture*. In Proceedings of the 20th annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '05). ACM, New York, NY, USA. Vol. 40, 10. pp. 167-176.
- [Sant'Anna et al. 2007] SANT'ANNA, C., FIGUEIREDO, E., GARCIA, A., LUCENA, C. *On the Modularity Assessment of Software Architectures: Do my Architectural Concerns Count?* In Proceedings of the 1st Workshop on Aspects in Architectural Description (AARCH) at the 6th International Conference on Aspect-Oriented Software Development (AOSD), Vancouver, Canadá.
- [Setamanit et al. 2007] SETAMANIT, S. et al. *Using simulation to evaluate global software development task allocation strategies*. Software Process: Improvement and Practice. Vol. 12, 5. pp. 491-503.
- [Shen et al. 2002] SHEN, M. ET AL. *Multi-Criteria Task Assignment in Workflow Management Systems*. In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, pp. 6-9.
-

- [Silva 2008] SILVA, C. A. *DiSEN-SCV Gerenciador de Versões de Artefatos para um Ambiente de Desenvolvimento Distribuído de Software*, Master Thesis, Universidade Estadual de Maringá.
- [Sooraj, Mohapatra 2008] SOORAJ, P., MOHAPATRA, P. K. J. *Developing an Inter-site Coordination Index for Global Software Development*. In Proceedings of the 2008 IEEE International Conference on Global Software Engineering, pp. 119-128.
- [Sosa, Eppinger et al. 2002] SOSA, M. E., S. D. EPPINGER, et al. *Factors that influence Technical Communication in Distributed Product Development: An Empirical Study in the Telecommunications Industry*. IEEE Transactions on Engineering Management. Vol. 49, 1. pp. 45-58.
- [Souza Filho et al. 2007] SOUZA FILHO, G. L. de, LEITE, L. E. C. e BATISTA, C. E. C. F. 2007. *Ginga-J: The Procedural Middleware for the Brazilian Digital TV System*. Brazilian Computer Society. Vol. 12, pp. 47-56.
- [Sommerville 2007] SOMMERVILLE, I. *Software Engineering*. 8th edition, Pearson Education Inc, Addison-Wesley.
- [Staudenmayer 1997] STAUDENMAYER, N. A. *Interdependency: Conceptual, Empirical, & Practical Issues*. Working paper, Sloan School of Management, Massachusetts Institute of Technology, Cambridge.
- [Sullivan et al. 2001] SULLIVAN, K. J., GRISWOLD, W. G., CAI, Y., and HALLEN, B. *The structure and value of modularity in software design*. In Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-9). ACM, NY, pp. 99-108.
- [Thebeau 2001] THEBEAU, R. E. *Knowledge management of system interfaces and interactions for product development processes*, Master of Science Thesis, MIT.
- [Vieira 2004] VIEIRA, M. R. E. *A Compositional Approach for Analyzing Dependencies in Component-Based Systems*. Information and Computer Science. Doctoral Thesis. University of California, Irvine.
- [Wang, Antonsson 2004] WANG, B., ANTONSSON, E. *Information measure for modularity in engineering design*. In Proceedings of the ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2004). Salt lake City, Utah, USA. Vol. 3a, pp. 449-458.
- [Whitfield et al. 2002] WHITFIELD, R. et al. *Identifying component modules*. In Proceedings of the 7th International Conference on Artificial Intelligence in Design (AID '02), pp. 571-592.
- [Yu et al. 2003] YU, T-L., YASSINE, A. A., GOLDBERG, D. E. *A Genetic Algorithm for Developing Modular Product Architectures*. In Proceedings of the ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2003). Chicago, Illinois, USA. Vol. 3b, pp. 515-524.
-

[Yu et al. 2007] YU, T.-L. et al. *An information theoretic method for developing modular architectures using genetic algorithms*. Research in Engineering Design. Springer London. Vol. 18, pp. 91-109.

[Zolfaghari et al. 2002] ZOLFAGHARI, S.; LIANG, M. *Comparative study of simulated annealing, genetic algorithms and tabu search of solving binary and comprehensive machine-grouping problems*. Production Research. Vol. 40, 9. pp. 2141-21
